Is Batch Size a Real Hyperparameter?

Alexey Tochin

2021-04-08

Table of contents

1 Counting Samples, Not Steps **3**
1.1 Rescaling the learning rate . . . 3
1.2 Experimental check . . . 6

2 The Noise: Batch Size as a Langevin Time Step **7**
2.1 The noise per step . . . 7
2.2 The Langevin algorithm . . . 8

3 Convergence Stability **11**

4 Gradient Descent with Momentum **14**

5 Why Adam Is Weird **18**
5.1 Adam : replacing the square of the mean . . . 19
5.2 The noise of a batch . . . 20
5.3 Where each algorithm puts the gate . . . 21
5.4 What a rescaling can and cannot do . . . 22
5.5 A variant that does scale . . . 24

6 Conclusion **25**
6.1 What This Means in Practice . . . 25
6.2 What This Does Not Cover . . . 26
6.3 A Word to the Reader . . . 26

7 Postscript **26**

References **27**

The trick that started this article is gradient accumulation. When the GPU runs out but we still want a larger batch, we split it into k micro-batches, run them one after another at the

same weight, sum the gradients and make a single step. The effective batch can then be as large as we like.

Accumulation buys none of the parallelism, since the micro-batches are sequential. It spends exactly the compute of k ordinary steps of the original size. The one thing it buys is a quieter step at the end, built from more gradients. But averaging gradients is momentum's job, and momentum does it without freezing the weights: all k backward passes move the model, where accumulation spends $k-1$ of them at a point we have already decided to leave.

This may sound obvious to the reader, which is why some of what follows reads as straightforward. The difficulty is turning it into an argument. To claim that a larger batch buys nothing, we have to say what stays fixed while it grows. With the textbook hyperparameters nothing does. Raise the batch size and leave the learning rate alone, and the training changes for a trivial reason: the steps got longer. So the question of the batch size is really a question about how gradient descent ought to be parameterized in the first place.

The neighbouring question is the practical one. Suppose we have tuned the learning rate and the other hyperparameters for a given model, and more powerful hardware then allows a larger batch. By how much should we rescale the ordinary learning rate to keep that tuning: linearly in the batch size, as its square root, or not at all? Otherwise the tuned setting quietly stops being tuned, and the larger batch that was supposed to come free costs another round of searching. The two questions have the same root, and Section 1 answers both at once.

With that in hand we can put the question to one optimizer at a time, in order of increasing resistance. Section 1 does the arithmetic for plain stochastic gradient descent and checks it against an experiment. Section 2 takes up the other half of the same claim, the noise the updates inject. There the batch size plays the part of a time step in the Langevin algorithm. Section 3 draws the boundary of the region where any of that applies, since none of it survives unstable training. Section 4 then adds momentum, where the batch size and the smoothing become two parameters with one job each. Section 5 reaches Adam, where the tidy picture stops working.

The evidence below is narrow. The experiments use MNIST with a small convolutional network, a constant learning rate and no weight decay. Batch normalization is deliberately absent: it couples the samples inside a batch, which is the effect we are trying to isolate. The article reasons about what ought to hold in principle and checks that on a small problem. In practice weight decay, normalization or training close to the stability boundary of Section 3 can each change the outcome, and Section 7 describes one case where a rule derived here does not survive at scale.

1 Counting Samples, Not Steps

1.1 Rescaling the learning rate

Let $\theta \in \mathbb{R}^D$ be the weight vector of a model with D trainable weights. We write the sequence of weights produced by training as $\{\theta_n\}_{n=0,1,2,3,...}$, starting from an initialization θ_0.

Let $L(x, y; \theta)$ be the loss function for a supervised model with weight θ, and a single data sample with feature x and label y. Define the loss gradient at point θ for sample (x_i, y_i) with index $i = 0, 1, 2, ...$ as

$$\mathbf{g}_i(\theta) := \frac{\partial}{\partial \theta} L(x_i, y_i; \theta). \tag{1}$$

The samples $\{(x_i, y_i)\}_{i=0,1,2,...}$ are grouped into batches. We consider a fixed batch size $b = 1, 2, 3, ...$ We define an integer-valued function t

$$t: \quad \{0, 1, 2, ...\} \quad \to \quad \{0, 1, 2, ...\} \tag{2}$$

that maps the index of each sample to the index of the batch it belongs to. For example, for batch size $b = 3$, we have

$$t_0 = 0,\ t_1 = 0,\ t_2 = 0,\ t_3 = 1,\ t_4 = 1,\ t_5 = 1,\ t_6 = 2,\ t_7 = 2,\ t_8 = 2,\ ... \ . \tag{3}$$

We begin with the simplest version of stochastic gradient descent without momentum, parameterized by the learning rate $\lambda^{\rm cl} > 0$. The weight update rule is

$$\theta_n \ \to \ \theta_{n+1} := \theta_n - \lambda^{\rm cl}\, b^{-1} \sum_{i:\ t(i)=n} \mathbf{g}_i(\theta_n), \quad n = 0,\ 1,\ 2,\ ... \tag{4}$$

where $\sum_{i:\ t(i)=n}$ ... is a sum over the minibatch n. The superscript " $^{\rm cl}$" marks the "classically normalized" hyperparameters that we are going to reconsider.

Suppose the batch size b changes. We want to adjust $\lambda^{\rm cl}$ so that the weight sequence $\{\theta_n\}_{n=0,1,2,3,...}$ moves as little as possible. More precisely, after the batch size change $b \to \widetilde{b}$, each of the weights $\widetilde{\theta}_{\widetilde{t}(i)}$ from the new sequence should be close to the original ones $\theta_{t(i)}$ for each $i = 0, 1, 2, ...$, where $\widetilde{t}$ corresponds to the new batch size $\widetilde{b}$. A reasonable adjustment is then

$$\lambda^{\rm cl} \to \widetilde{\lambda}^{\rm cl} = \frac{\widetilde{b}}{b}\, \lambda^{\rm cl}. \tag{5}$$

Equivalently, the product $\lambda^{\rm cl} b^{-1}$ is left unchanged by this rescaling, which makes it the natural variable. We give it a name:

$$\lambda := \lambda^{\rm cl}\, b^{-1}, \qquad \text{equivalently} \qquad \lambda^{\rm cl} = b\, \lambda. \tag{6}$$

The explicit factor b^{-1} in (4) is then absorbed into λ, and the update rule becomes

$$\theta_{n+1} = \theta_n - \lambda \sum_{i:\ t(i)=n} \mathbf{g}_i(\theta_n). \tag{7}$$

Each sample now contributes $-\lambda\,\mathbf{g}_i$ to the weight update, whatever batch it happens to fall in.

Unrolling (7) from the initialization says the same thing in closed form,

$$\theta_{n+1} = \theta_0 - \lambda \sum_{i:\ t(i)\le n} \mathbf{g}_i\left(\theta_{t(i)}\right), \tag{8}$$

one term per sample consumed and no b in sight. The batch size has not vanished from the problem. It survives in the arguments $\theta_{t(i)}$, which record at which weight each sample's gradient was taken, and a larger batch means more samples sharing one argument. It no longer multiplies anything. λ is a common factor of the whole displacement, while b decides only where in the trajectory the gradients are evaluated.

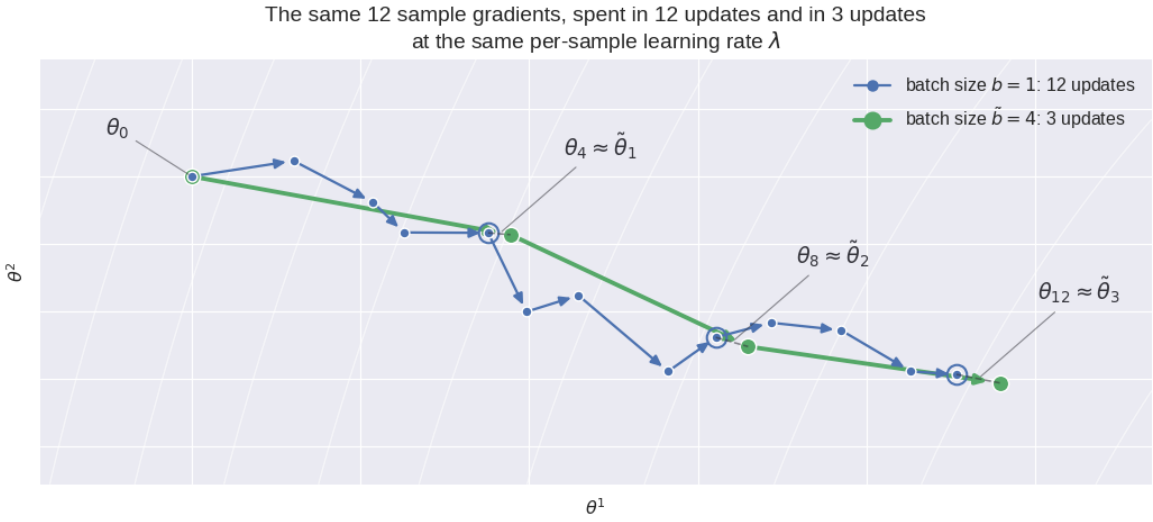

Figure 1: Two runs over the same 12 sample gradients, taken in the same order and with the same per-sample learning rate λ. The first one spends them in 12 updates with the batch size $b = 1$, the second one in 3 updates with the batch size $\tilde{b} = 4$. Every fourth weight of the first chain is matched by one weight of the second one, and the two disagree only because the gradients inside a batch are evaluated at a single weight rather than at the 4 nearby ones the first chain passes through.

Read in the classic variables, (6) is the familiar linear scaling rule $\lambda^{\mathrm{cl}} \propto b$, standard practice in large-batch training at least since Goyal et al. trained ImageNet in an hour. Krizhevsky

reported it as a heuristic, and recorded a puzzle with it: multiplying the learning rate by k worked best in his experiments, while theory, he wrote, asked for $\sqrt{k}$ "to keep the variance in the gradient expectation constant", and he could not explain the discrepancy. We can explain the discrepancy. The $\sqrt{k}$ rule holds the noise of one update fixed, which Section 2 shows is not what stays fixed over a budget of samples. His footnote that the heuristic "does eventually break down for batch sizes larger than the ones considered" is the restriction of Section 3.

In the per-sample variables there is no rule to apply: once b no longer stands next to the learning rate, there is nothing left to rescale by hand.

For example, raise the batch size by a factor k. The k sequential steps become a single step over k times as many samples. The two sums differ only in where their gradients are taken: at one weight in the large batch, and at k nearby weights in the small one, as Figure 1 shows.

If the batch size b equals the entire dataset size N or is of the same order of magnitude $b \lesssim N$, each update moves the weights down the loss of the whole dataset. In mini-batch training with $b \ll N$ this fails, because a single mini-batch is a very poor sample of the dataset. Mini-batch learning works because those poor per-batch statistics average out over many updates: the signal accumulates across many samples and is not drawn from any single step.

Now take two runs at different batch sizes. Let $\theta_{t(i)}$ and $\tilde{\theta}_{\tilde{t}(i)}$ be the weights after i samples have been processed, in two runs with batch sizes b and $\tilde{b}$ at the same λ. For small λ, large i and finite b,

$$\left\|\tilde{\theta}_{\tilde{t}(i)} - \theta_{t(i)}\right\| = O\left(\lambda^2\, i\right). \tag{9}$$

Both powers can be read off (8). The displacement itself is λ times a sum of i gradients, one factor of λ and i terms, so it is of order $\lambda\, i$. The difference between the two runs holds those same i gradients with the same factor λ in front, and differs only in the arguments. Each gradient is taken at the weight its own batch started from, and those weights lie less than one batch apart, a distance $O(\lambda)$ at a fixed batch size. Every term of the difference therefore carries two factors of λ, one from the rate and one from the lag, against the single factor the displacement carries, with the same i terms either way. Counting the mismatches once each keeps this a first-order statement: once the two runs have separated they begin to feed on that separation, which is why Section 1.2 compares losses over a long run rather than weights.

The two runs thus differ by a fraction of order λ of the distance they have both covered. Expanded in λ, the first-order term of the displacement is the same for every batch size, and b enters only at second order, which Section 1.2 tests.

No other rescaling of the hyperparameters with the batch size achieves (9). In (8) the per-sample rate multiplies the whole displacement, so changing it along with the batch size shifts the weights at first order in λ instead of at second. The batch size is then left with nothing to do but fill the hardware, and without momentum nothing does the smoothing either.

In mini-batch learning the natural control variable is the number of samples processed, the measure of the computation spent, rather than the number of updates performed. At a fixed λ

the weight trajectory is the same function of the number of samples at every batch size, to first order in λ, and λ is the only hyperparameter whose equality brings that about.

1.2 Experimental check

We test the reasoning above on MNIST with a simple convolutional network.

Model details

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape(input_shape=(28, 28, 1), target_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(kernel_size=3, filters=12, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=24, padding='same', strides=2, activation
    tf.keras.layers.Conv2D(kernel_size=6, filters=32, padding='same', strides=2, activation
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(units=10, activation=None)
])
```

The loss function is the standard classification cross-entropy.

In practice, comparing the weight update sequences directly is difficult. Instead, we consider the entire training progress and the mean loss convergence rate. The model does not overfit appreciably, so we report only training-set means below.

The experiment in Figure 2 is a grid of seven batch sizes $b = 1, 4, \dots, 4096$ against three learning rates $\lambda = 10^{-2}$, 10^{-3}, 10^{-4}, with five seeds for each pair. If the reasoning above is right, the curves for one λ must lie on top of each other whatever the batch size. They do wherever the training is stable: at a fixed λ the convergence rate does not depend on the batch size. The same invariance has been checked at a larger scale by Jastrzębski et al., who raise the batch size and the learning rate by a common factor on ResNet56 and VGG11 and find the learning curves matching when drawn against epochs. They also report that it matches more closely than raising the learning rate by the square root of that factor instead, which is the alternative the argument above rules out. Their networks carry batch normalization, so the loss itself depends there on how the samples are grouped, a complication absent from the runs above.

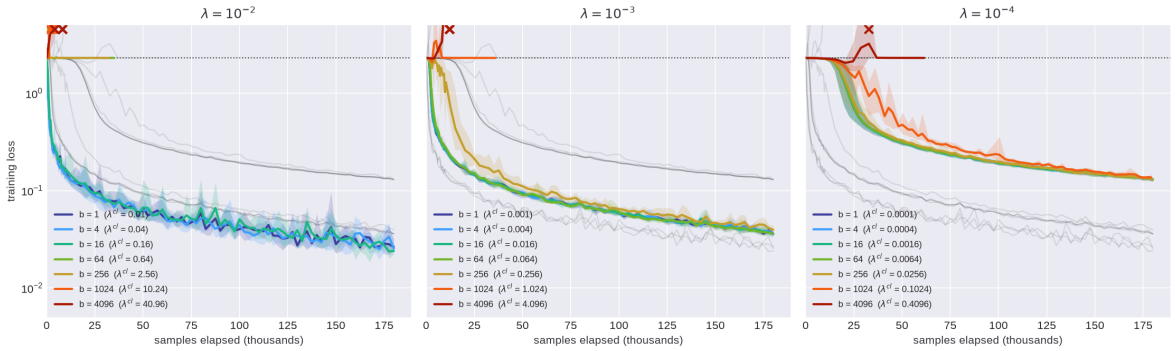

Figure 2: Training loss on MNIST against the number of elapsed samples, one panel per λ. The batch size runs along the rainbow, from $b = 1$ in indigo to $b = 4096$ in dark red. Each curve is the geometric mean over the seeds, the band is their spread, a cross marks a run that diverged, and the dotted line is the chance level $\ln 10 = 2.30$. The grey curves repeat the other panels, so the groups can be placed against each other, and they separate. The first batch size to fail is $b = 64$ at $\lambda = 10^{-2}$, $b = 1024$ at $\lambda = 10^{-3}$ and $b = 4096$ at $\lambda = 10^{-4}$, so a smaller λ tolerates a larger batch. Section 3 discusses the restriction.

2 The Noise: Batch Size as a Langevin Time Step

2.1 The noise per step

Section 1 settled what happens over a budget of samples: to first order in λ, the accumulated update is the same sum of the same gradients, only grouped differently, so nothing about it can depend on b. The batch size does change the individual step. A larger batch makes fewer and longer steps, each carrying its own share of the noise, and it is not obvious that a different pattern of noisy steps leads to the same place.

Why check the noise at all, once the drift is settled? Because in deep learning the noise helps decide where the training ends up, and is no small correction to the drift. Gradient descent does not settle at a distinguished point. It keeps moving through a region of low loss, and it stops where the sample budget runs out. The trained network is then closer to a single draw from a distribution over weights than to the solution of an optimization problem, and that distribution is shaped by the noise as well as by the drift. Training behaves like a random sampler stopped after producing a single sample. As a first picture, take the distribution it samples from to be

$$\pi(\theta) \propto e^{-\alpha L(\theta)}, \tag{10}$$

where $L(\theta)$ is the loss averaged over the N training samples and $\alpha > 0$ is a constant. The constant α sets how strongly the sampler prefers a low loss. For a large α the distribution concentrates where L is smallest, and as $\alpha \to \infty$ sampling turns into plain minimization. For

a small α it spreads over a wide region of weights where the loss is only moderately low. In the language of statistical physics α is an inverse temperature. One value is special: for the cross-entropy loss used here, $e^{-NL(\theta)}$ is the likelihood of the training data, so $\alpha = N$ gives the Bayesian posterior with a flat prior. Training is not bound to that value. Nobody sets α directly; it is fixed by the noise of the training, and the rest of this section works out which hyperparameter controls it. Mandt, Hoffman and Blei work this view out in detail. Dropout, which our example network uses, adds noise of its own to that of the samples. Incidentally, under certain conditions training with dropout is itself approximate Bayesian inference (Gal and Ghahramani), each random mask drawing one network from an approximation to the posterior.

> **i** A caveat: reshuffling once per epoch
>
> The sampler picture, and the comparison with the Langevin algorithm below, treat the noise of different updates as independent. The independence is exact when every batch is drawn afresh from the data. When the data are instead reshuffled once per epoch, the batches of one epoch partition the dataset, and at a frozen weight their gradients add up to the full-batch gradient. The noise of the sampling cancels, and only that of the dropout masks, drawn afresh at every step, remains. On the scale of an epoch and longer the sampler picture then overstates the noise, by an amount that depends on how far the weights move within the epoch. None of this touches the invariance of Section 1, which holds for any fixed order of the data.

We will not need that theory, only its simplest consequence: the fluctuations are part of what training computes, so a reparameterization that calls the batch size technical has to leave them alone as well as the drift. This section asks whether it does.

Let $\bar{\mathbf{g}} \in \mathbb{R}^D$ and the $D \times D$ matrix Σ be the mean and the covariance of a single-sample gradient $\mathbf{g}_i$ at a given weight, over the draw of the sample:

$$\bar{\mathbf{g}} := \mathbb{E}_i\left[\mathbf{g}_i\right], \qquad \Sigma := \mathbb{E}_i\left[\left(\mathbf{g}_i - \bar{\mathbf{g}}\right)\left(\mathbf{g}_i - \bar{\mathbf{g}}\right)^{\mathsf{T}}\right]. \tag{11}$$

For independently drawn samples, one update (7) moves the weight by a random amount with

$$\mathbb{E}\left[\Delta\theta\right] = -\lambda\, b\, \bar{\mathbf{g}}, \qquad \mathrm{Cov}\left[\Delta\theta\right] = \lambda^2\, b\, \Sigma, \tag{12}$$

the drift adding up over the b samples and the covariance adding up too, since the cross terms vanish. At a fixed λ both are proportional to b, so the random kick goes like $\sqrt{b}$.

2.2 The Langevin algorithm

Suppose we want to draw random points from a given distribution over the weights,

$$\pi(\theta) \propto e^{-U(\theta)/T}, \tag{13}$$

with an energy U and a temperature T. With $U = L$ and $T = 1/\alpha$ this is (10). Drawing such points is the task of the random sampler we compared training to above, and the task of Bayesian inference too, with π the posterior. One way to do it is to iterate

$$\theta_{n+1} = \theta_n - h\,\nabla U(\theta_n) + \sqrt{2\,T\,h}\;\xi_n, \tag{14}$$

with ξ_n independent standard normal vectors and a small step h. After many iterations the points are distributed approximately according to π, the more accurately the smaller the step. This iteration is known as the unadjusted Langevin algorithm (Roberts and Tweedie). Run with a stochastic gradient in place of ∇U, it becomes stochastic gradient Langevin dynamics (Welling and Teh), in which the noise is injected into gradient descent deliberately. The algorithm is a step-by-step approximation of a process in continuous time, the overdamped Langevin equation

$$d\theta = -\nabla U(\theta)\,dt + \sqrt{2\,T}\;d\mathbf{W}_t, \tag{15}$$

in which the weight slides down U while being pushed around by a random force $\mathbf{W}_t$ whose accumulated effect has a variance growing in proportion to the elapsed time. The approximation is the simplest one possible. Over a short time h it keeps the downhill pull fixed at its value at the start of the step, and it replaces the random pushes by a single normal kick with the variance they would have accumulated over that time, $2Th$. (This recipe is known as the Euler–Maruyama scheme.) The square root in front of the noise in (14) is what that recipe dictates: the drift over a step is proportional to h while the kick is only of size $\sqrt{2Th}$. The asymmetry is the rule a change of step has to follow if the algorithm is to keep sampling the same π: scale the noise any other way, and changing h changes the temperature, and with it the distribution. The step itself is a trade. A larger h covers ground faster, and pays with a bias that grows with the step and, once the step is too long for the curvature of U, with the loss of convergence altogether.

To compare the two algorithms, write the gradient descent update (7) in the same shape as (14). Split every single-sample gradient into its mean and a random remainder, $\mathbf{g}_i = \bar{\mathbf{g}} + (\mathbf{g}_i - \bar{\mathbf{g}})$. The update becomes

$$\theta_{n+1} = \theta_n - \lambda b\;\bar{\mathbf{g}}(\theta_n) + \sqrt{\lambda b}\;\zeta_n, \qquad \zeta_n := -\sqrt{\lambda/b}\sum_{i:\;t(i)=n}\left(\mathbf{g}_i(\theta_n) - \bar{\mathbf{g}}(\theta_n)\right), \tag{16}$$

where the random vector ζ_n has mean zero and, by (12), covariance $\lambda\Sigma$. Now (14) and (16) can be read term by term:

	Langevin (14)	gradient descent (16)
step	h	λb
downhill direction	∇U	$\bar{\mathbf{g}} = \nabla L$
noise covariance per unit of step	$2T \cdot I$	$\lambda\Sigma$

Here I is the $D \times D$ identity matrix. The last row is where the analogy is loose. The Langevin noise is the same in every direction, so its covariance is a multiple of the identity, while Σ is a general matrix that also changes with the weights. Apart from that, the two updates have the same form, and the $\sqrt{b}$ growth of the random kick found above is the $\sqrt{h}$ of the Langevin algorithm. A change of b at a fixed λ changes h and nothing else. The batch size is the time step of the discretization, and $\lambda\Sigma$ is its temperature. Whatever distribution the iteration settles into is fixed by $\bar{\mathbf{g}}$ and by that temperature; b reaches it only through the step, and so only through the corrections that any discretization carries. Changing λ is a different matter, since it rescales the temperature and therefore the distribution itself. So the batch size is technical not only for the deterministic part of the update but for its noise as well: it sets the resolution at which one and the same random process is followed. Stochastic equations of this kind have been derived for gradient descent systematically, as an expansion in the learning rate, by Li, Tai and E. Jastrzębski et al. make the same identification. They single out the ratio of the learning rate to the batch size, which is λ here, as the quantity the dynamics depend on. Two runs that share it, they argue, are two discretizations of one stochastic differential equation.

The looseness in the last row of the table has a price. The noise of gradient descent comes from the data rather than being injected, and its covariance Σ depends on the weights and is far from isotropic. The distribution that the iteration (16) settles into, where there is one, is therefore in general not (10), which it would be if Σ were a constant multiple of the identity. $\lambda\Sigma$ is a temperature in the looser sense that it sets the size of the fluctuations, and of its two factors only λ is ours to choose, Σ being fixed by the model and the data. When we call λ the temperature below, we mean this controllable part.

So far we have compared only two moments. For a large batch the central limit theorem gives more: ζ_n is a sum of b independent centred terms, so its distribution is close to normal, and (16) takes the form of (14) exactly, the $\sqrt{\text{step}}$ scaling of the noise included. What the comparison gives is the form of the iteration and not its stationary law: the noise covariance per unit of step is $\lambda\Sigma$ where the Langevin algorithm has $2T \cdot I$, which is the looseness of the last row of the table, and no batch size touches it. The central limit theorem does not hand over Σ either. It is a property of the model and the data, and we neither choose it nor, in practice, compute it. The analogy says how the fluctuations must change when the step changes, and that their size is governed by λ, but it does not say what that size is. It does explain what mini-batch practice keeps running into: too small a learning rate gives a colder search that explores less.

This agrees with the observations of Samuel L. Smith and Quoc V. Le. They report that "…optimum batch size is proportional to the learning rate", which says, in our variables, that there is an optimal λ and that it does not depend on b. The familiar trade-off is real, but it is a trade-off in λ, not in b: a large λ means a hot, strongly fluctuating search that generalizes well but settles poorly, while a small λ gives a cold search that settles to a lower training loss. Claims of the form "small batches generalize better" are claims about a fixed λ^{cl}, where shrinking b raises $\lambda = \lambda^{\text{cl}}/b$, and with it the temperature. Once λ is the control variable, the trade-off is tuned by changing λ, and the batch size plays no part in it.

3 Convergence Stability

Most of the statements of this article carry the caveat “if the training is stable”, so the topic cannot be avoided altogether. We measure where that region ends for the problem at hand, say what kind of restriction each of the two boundaries is, and stop there.

Gradient descent becomes unstable once the learning rate passes a certain value. For a classical optimization problem, where the batch is the entire dataset, the threshold belongs to the deterministic step. Mini-batch training is less simple. A step can be read as a deterministic step plus noise, the way (16) writes it, and a single mini-batch is no more informative than the samples inside it. So we expect the threshold to be set by the two per-sample quantities of Section 1 and Section 2 rather than by the step alone.

Figure 3 shows the result for the MNIST problem introduced above.

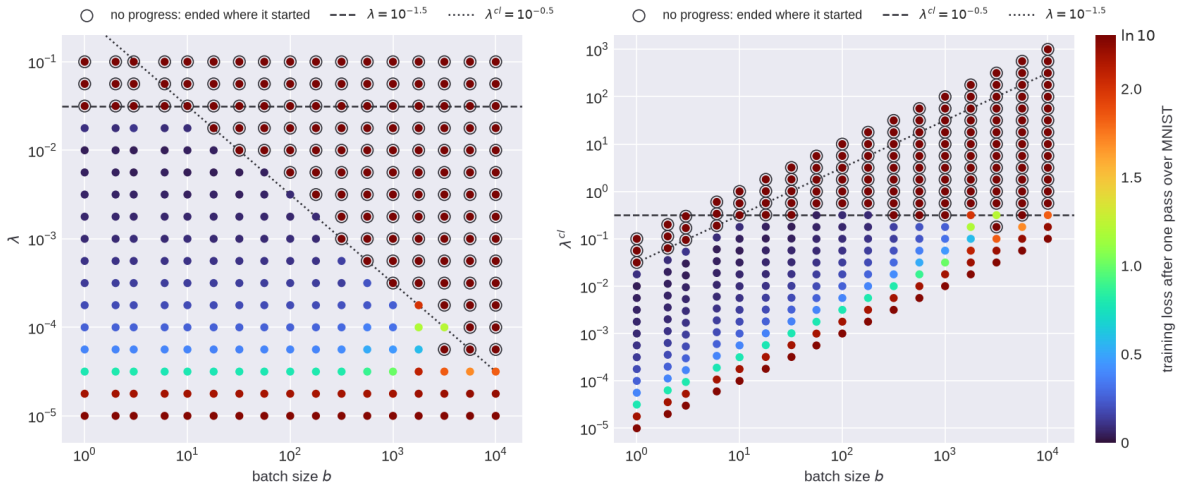

Figure 3: Each dot is one training run of plain gradient descent on MNIST, placed at its batch size and learning rate and coloured by the training loss at the end. Every run is given the same budget of 60,000 samples, one pass over the dataset, and not the same number of updates. Dark red is the chance level $\ln 10 = 2.30$ for the ten MNIST classes, where a run that blew up is reported as well. A ring marks a run that ended within one percent of the loss it started from, which separates a run that was lost from one that was merely slow: at the smallest λ the budget runs out with the loss still high, but it has come down, and the dot carries no ring. The two panels hold the same runs under the two normalizations of the learning rate, see (6). In each panel the dashed line is the restriction on the variable of that panel, and the dotted line the restriction on the other one.

The lost region is bounded by two straight lines, and both conditions have to hold at once:

1. $\lambda^{\mathrm{cl}} = \lambda\, b \lesssim 10^{-0.5}$,
2. $\lambda = \lambda^{\mathrm{cl}}/b \lesssim 10^{-1.5}$.

The first is the classical restriction. Within one update the weight moves by λ^{cl} times a batch-averaged gradient, and that displacement has to stay small against the scale on which the mean gradient $\bar{\mathbf{g}}$ itself varies, the scale set by the curvature of L along the trajectory. Too long a step overshoots, exactly as in deterministic optimization, and nothing about this is particular to mini-batches.

The second is less banal. What carries the weights away is the spread of the gradient from one sample to the next, not the curvature of the mean. The parameter that measures it is λ, the temperature of Section 2, and not λ^{cl}.

Two observations support reading the two boundaries as restrictions of different kinds. The first is the difference between the way a run is lost at $b = 1$ and at $b = 1000$.

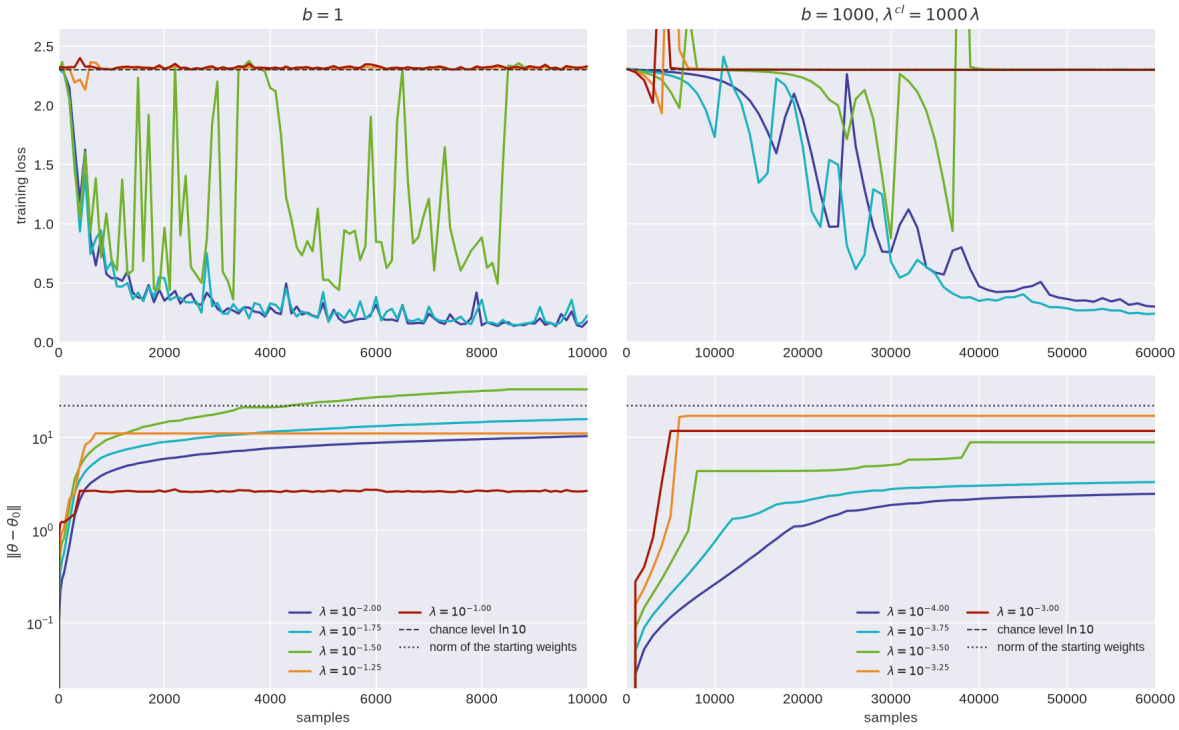

Figure 4: The training loss (top) and the Euclidean distance of the weights from where they started (bottom), for five learning rates at each of two batch sizes, taken from the grid of Figure 3. Each column covers the range that settles its own outcome: 10,000 samples at $b = 1$, which is 10,000 updates, and the whole pass at $b = 1000$, which is 60. In both columns the two smallest rates train and the three largest are lost. The dashed line is the chance level and the dotted line the norm of the starting weights, $\|\theta_0\| = 22.5$.

At $b = 1000$ a lost run leaves within a few updates and then stops. The loss sits at the chance level and $\|\theta - \theta_0\|$ holds its value to four digits for the rest of the pass: the network has reached a state in which almost every rectifier is off, which has no gradient and no way back. The same runs with a leaky rectifier, which has no such state, blow up instead of freezing. At $b = 1$ nothing stops. At the largest rate traced, $\lambda = 10^{-1}$, the weights keep wandering by a few percent of the distance they have covered while the loss stays at chance.

The second observation raises the noise by hand. We add an independent Gaussian of standard deviation κ to the gradient of every sample, which leaves the mean gradient where it was and widens the spread around it. That noise is isotropic, where the sampling noise is not, so the experiment raises the noise level without imitating the sampling. If the second restriction is a restriction on noise, it has to come down as κ grows, while the first stays where it is. Figure 5 shows that this is what happens.

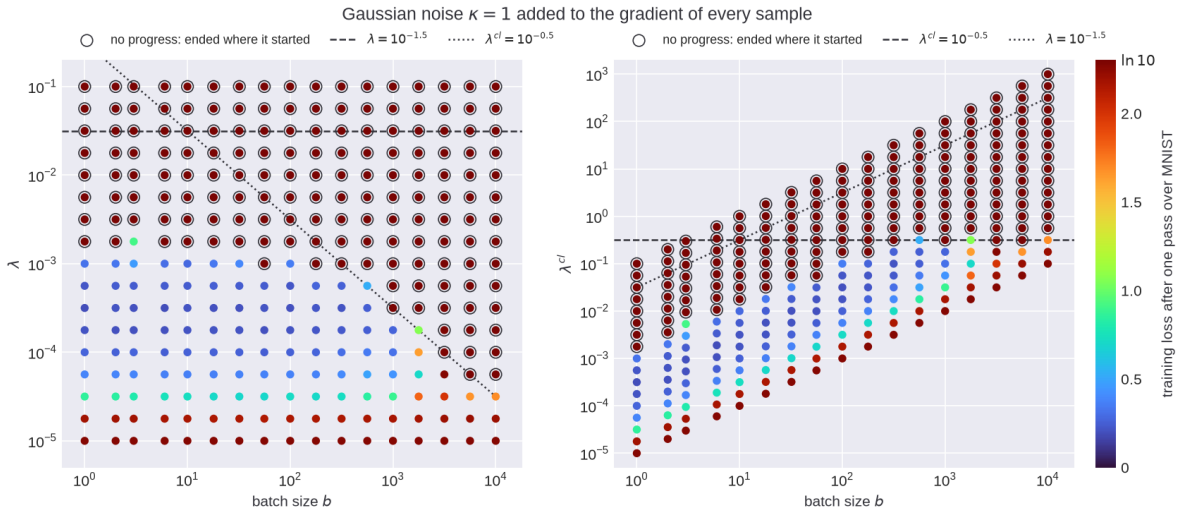

Figure 5: The grid of Figure 3 run again with noise added by hand. Every single-sample gradient becomes $\mathbf{g}_i + \kappa\,\xi_i$, with ξ_i an independent standard normal vector drawn afresh for every weight and every sample, at $\kappa = 1$. The batch mean then carries an extra noise of standard deviation $\kappa/\sqrt{b}$ per weight, which is the batch-size dependence the sampling noise has as well. This κ is far above the sampling noise of the problem itself, whose standard deviation at θ_0 is $2.1 \cdot 10^{-3}$ per weight, so the injected noise dominates everywhere in the panel. The dashed and dotted lines are the two restrictions of Figure 3, left where they were: the restriction on λ has dropped by more than a decade, to about 10^{-3}, while the one on λ^{cl} has not moved.

Both thresholds depend on the model and the data. In practice it is the second that the batch size runs into first, and that is an argument for the (λ, b) parameterization over the classical $(\lambda^{\mathrm{cl}}, b)$.

A ceiling of this kind is well known in large-batch training under the name *critical batch size*.

Shallue et al. measure it across many models and datasets: the progress per sample stays flat as the batch grows, exactly as in Figure 2, and then degrades beyond a problem-dependent threshold. McCandlish et al. predict where that threshold sits from the gradient noise.

The subject raises more questions than it answers, and the stability of training is not what this article is about, so we stop here.

4 Gradient Descent with Momentum

We now add momentum. Its most widely used parameterization is

$$\begin{aligned} \mathbf{v}^{\mathrm{cl}}_{n+1} &= m^{\mathrm{cl}}\,\mathbf{v}^{\mathrm{cl}}_{n} - \lambda^{\mathrm{cl}}\,b^{-1} \sum_{i:\ t(i)=n} \mathbf{g}_i(\theta_n) \\ \theta_{n+1} &= \theta_n + \mathbf{v}^{\mathrm{cl}}_{n+1} \end{aligned}, \quad 0 \le m^{\mathrm{cl}} < 1, \quad \lambda^{\mathrm{cl}} > 0 \tag{17}$$

The parameter $\mathbf{v}^{\mathrm{cl}}$ has a well-known physical reading. Take n for a discrete time, and $\mathbf{v}^{\mathrm{cl}}$ is a velocity in the space of weights, with the batch-averaged gradient as the acceleration. We use a different aspect of momentum here.

The exponential moving average for the sequence $\{g_n\}_{n=0,1,2,\dots}$ is a sequence $\{s_n\}_{n=0,1,2,\dots}$ defined by the recursion

$$s_{n+1} = m\,s_n + (1-m)\,g_n, \quad 0 \le m < 1 \tag{18}$$

with some initial condition for s_0 that we are not interested in. Unrolling it,

$$s_{n+1} = (1-m) \sum_{k=0,1,2,\dots} m^k\,g_{n-k}. \tag{19}$$

So momentum smooths the gradients of previous steps. Instead of using only the gradients at the current weight θ_n, we sum the gradients taken at all earlier weights θ_{n-1}, θ_{n-2}, θ_{n-3}, and so on, with the decreasing coefficients $(1-m)m$, $(1-m)m^2$, $(1-m)m^3$. The coefficients sum to one:

$$(1-m) \sum_{k=0,1,2,\dots} m^k = 1, \quad 0 \le m < 1. \tag{20}$$

The same formula shows where momentum and gradient accumulation genuinely differ, and why the introduction's claim that momentum already does accumulation's job holds only for small λ. Accumulation averages the gradients of its micro-batches at one weight, the current one. Momentum averages gradients taken at earlier weights, stale by about one smoothing time, over which the weights move a distance of order λ. The difference is exactly that displacement, and it is negligible in the same regime in which the $O(\lambda)$ corrections of Section 1 can be dropped.

The parameter m sets how fast those coefficients decay. We want this rate fixed per sample, as in Section 1. The coefficient multiplying each single-sample gradient g_i should change as little as possible when the batch size changes, which gives

$$\sqrt[b]{m} = \sqrt[\tilde{b}]{\tilde{m}}. \tag{21}$$

So the momentum must be rescaled as

$$m \to \tilde{m} = m^{\frac{\tilde{b}}{b}} \tag{22}$$

to keep the smoothing per sample the same. Figure 6 shows what this means.

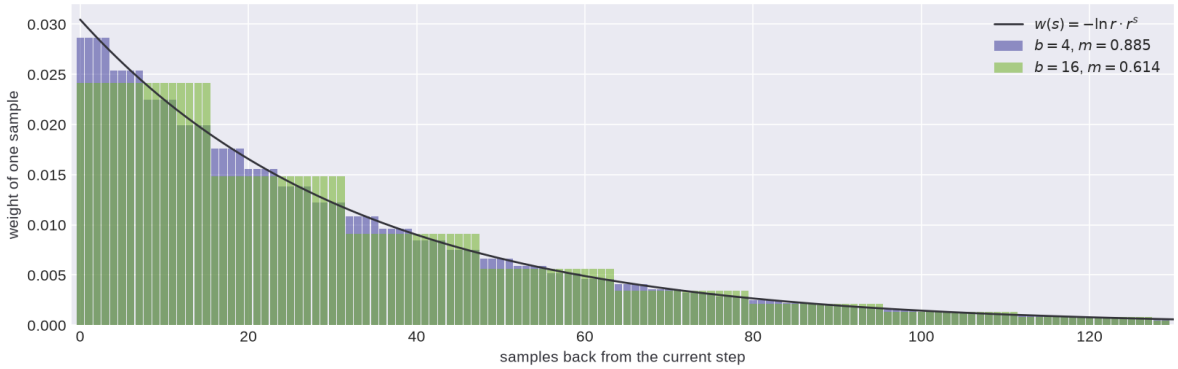

Figure 6: Weight coefficients with which each sample gradient g_i contributes to a training step under the condition (21). The batch sizes are 4 and 16, and $\sqrt[b]{m} = \sqrt[\tilde{b}]{\tilde{m}} = 0.97$, so that each sample retains, on average, 97% of the weight of the previous one, which corresponds to $m = 0.885$ at $b = 4$ and $\tilde{m} = 0.614$ at $\tilde{b} = 16$. The coefficients are grouped into blocks of $b = 4$ or $\tilde{b} = 16$, one block per batch. At either batch size the momentum can be tuned so that the coefficients nearly agree. The curve is the same weighting with the batch taken to a single sample, $w(s) = -\ln r \cdot r^s$ with $r = 0.97$, and each block sits at the average of that curve across it.

The goal of the reparameterization is to separate the technical part, data parallelism, from the real one, the smoothing between gradients. This motivates an alternative parameterization of (17):

$$\begin{aligned} \mathbf{v}_{n+1} &= e^{-\frac{b}{\tau}}\, \mathbf{v}_n + (1 - e^{-\frac{b}{\tau}}) \sum_{i:\, t(i)=n} \mathbf{g}_i(\theta_n) \\ \theta_{n+1} &= \theta_n - \lambda\, \mathbf{v}_{n+1} \end{aligned} \quad , \qquad \tau \geq 0, \quad \lambda > 0. \tag{23}$$

The new parameter τ is defined as

$$\tau := -\frac{b}{\ln m^{\mathrm{cl}}}, \quad \text{equivalently} \quad m^{\mathrm{cl}} = e^{-\frac{b}{\tau}}, \tag{24}$$

so that $e^{-\frac{1}{\tau}} = \sqrt[b]{m^{\mathrm{cl}}}$ is exactly the per-sample decay rate that (21) asks to be kept fixed. τ is the smoothing length measured in samples: each sample gradient loses the factor $e^{-\frac{1}{\tau}}$ of

its weight per elapsed sample, so its contribution decays by e over τ samples regardless of the batch size. We read $\tau = 0$ as the limit $\tau \to 0^+$, where (23) degenerates to plain gradient descent, (4) with $m^{\mathrm{cl}} = 0$. The larger the value of τ, the stronger the smoothing.

The advantage is that moving to different hardware does not call for retuning λ and τ. We set the batch size b as large as possible, or as large as the stability restrictions of Section 3 allow. Figure 7 confirms this experimentally.

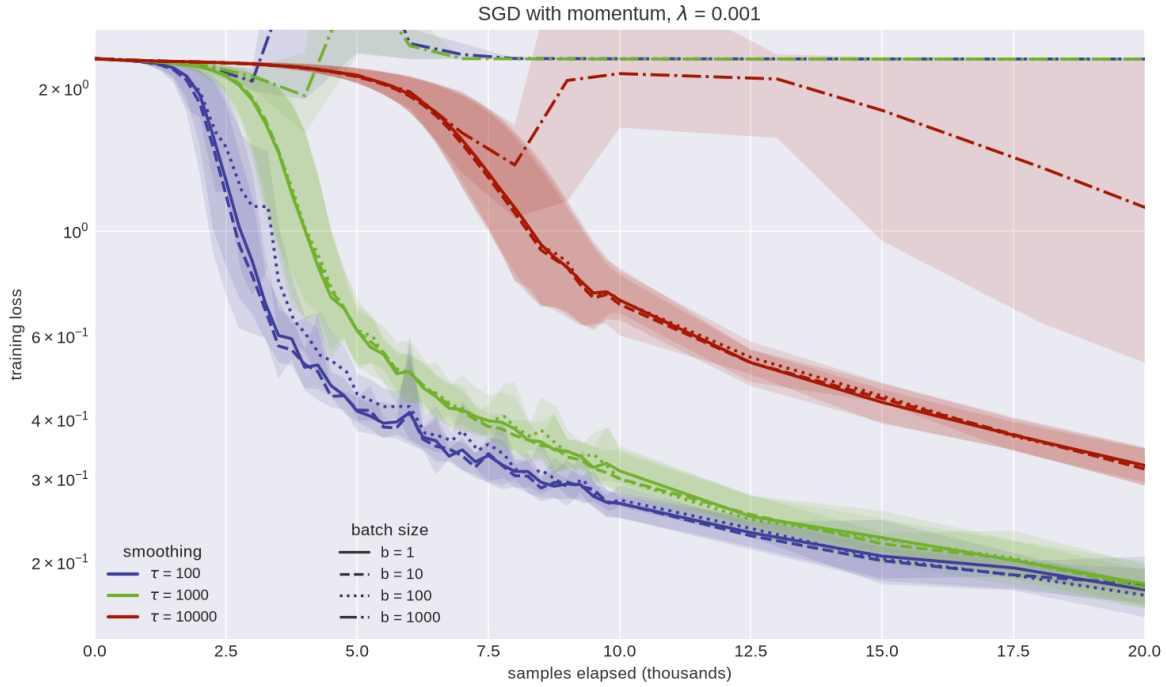

Figure 7: Training loss on MNIST for (23) at $\lambda = 10^{-3}$. Each curve is the geometric mean over 5 seeds, and the band is their spread. Before each run we warm the velocity up over $10\,\tau$ samples at $\lambda = 0$, so the smoothing is already full when the weights start to move. At fixed λ and τ the progress per sample does not depend on the batch size, until stability is lost between $b = 10^2$ and $b = 10^3$, where $\lambda^{\mathrm{cl}} = \lambda b$ leaves the range of Section 3. Those runs stay at chance level or leave the top of the panel. A longer τ converges more slowly here, which may differ for other problems.

For $b \gtrsim \tau$, however, the momentum smoothing is weak compared with the batch smoothing, and the approximation of Figure 6 no longer holds.

The relations between the proposed and the classic normalization are:

$$\begin{aligned} m^{\mathrm{cl}} &= e^{-\frac{b}{\tau}} \\ \lambda^{\mathrm{cl}} &= \left(1 - e^{-\frac{b}{\tau}}\right) \lambda\, b \end{aligned} \tag{25}$$

A rule of this kind is already in use. Smith, Kindermans, Ying and Le characterize momentum gradient descent by what they call the noise scale: the coefficient of the noise correlation in a Langevin equation of the kind Section 2 discusses, and a temperature in the sense used there. They prescribe holding it fixed when the batch size is changed. Their learning rate, batch size and momentum coefficient are λ^{cl}, b and m^{cl} here, so with N the size of the training set their quantity reads

$$\text{noise scale} := \frac{\lambda^{\mathrm{cl}}\, N}{b\,(1-m^{\mathrm{cl}})} = \frac{\left(1-e^{-\frac{b}{\tau}}\right)\lambda\, b\, N}{b\left(1-e^{-\frac{b}{\tau}}\right)} = \lambda\, N, \tag{26}$$

where the middle expression substitutes the two relations above. The factor N is theirs, carried by a cost summed over the training examples rather than averaged over them. So their prescription is the instruction to keep λ fixed, and neither the batch size nor the momentum coefficient enters it on its own.

That leaves τ free, which is where the two jobs separate. Holding λ fixed says nothing about the smoothing, and keeping the smoothing fixed as well is the extra demand of (22): raising the batch by a factor k asks for $m^{\mathrm{cl}} \to (m^{\mathrm{cl}})^{k}$, a momentum coefficient moving down. The cited authors move it up instead, on purpose, since their aim is to spend fewer updates rather than to keep the trajectory. They report the price: a slight loss of test accuracy, which they attribute to the averaging timescale growing until the average can no longer follow the training. They count that timescale in epochs; multiplied by the number of samples in an epoch it becomes

$$\text{averaging timescale} := \frac{b}{1-m^{\mathrm{cl}}} = \tau\left(1+O\left(\frac{b}{\tau}\right)\right) \tag{27}$$

samples, so what they are describing is a change of τ.

The effect of momentum itself is of the same order as the effect of the batch size, and the two come from the same place. A gradient enters the update at a weight the run has already left: by up to b samples inside a batch, and by about τ samples under the smoothing. Both corrections are of first order in λ with those lengths multiplying it, so at a small enough λ neither one is visible.

Yuan, Ying and Sayed proved this in the classical variables: momentum gradient descent with a small constant step size follows the trajectory of the plain method run at the step size

$$\frac{\lambda^{\mathrm{cl}}}{1-m^{\mathrm{cl}}} = \frac{\left(1-e^{-\frac{b}{\tau}}\right)\lambda\, b}{1-e^{-\frac{b}{\tau}}} = \lambda\, b, \tag{28}$$

which is the classical learning rate of plain gradient descent at the same per-sample rate, (6). Their rescaling therefore asks for nothing in the per-sample variables, and τ leaves the leading order. Li, Tai and E reach the same point from their stochastic equation for momentum, where the velocity relaxes over τ samples and then follows the gradient. The same combination carries the momentum in the noise scale of Smith and Le. Kidambi et al. go further and find no advantage in momentum on stochastic gradients at all, reading the gains seen in practice as a by-product of mini-batching.

We take the effects of momentum no further. The smoothing and the batch size differ in the length of the window they average over, then, and not in the order of magnitude in λ at which they enter, and this section needs only that the smoothing has a parameter of its own. Without momentum there is no such parameter and the smoothing simply goes undone. The batch size does not take it over: by Section 1 and Section 2 a larger batch buys a cleaner gradient per step and proportionally fewer steps, which leaves the noise over a budget of samples where it was.

5 Why Adam Is Weird

The same treatment applies to other methods. Adam in its classic parameterization reads, in our notation,

$$\begin{aligned}
\mathbf{v}^{\mathrm{cl}}_{n+1} &= \beta_1^{\mathrm{cl}}\,\mathbf{v}^{\mathrm{cl}}_{n} + (1-\beta_1^{\mathrm{cl}})\,b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i(\theta_n),\\
\mathbf{w}^{\mathrm{cl}}_{n+1} &= \beta_2^{\mathrm{cl}}\,\mathbf{w}^{\mathrm{cl}}_{n} + (1-\beta_2^{\mathrm{cl}})\left[b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i(\theta_n)\right]^2,\\
\theta_{n+1} &= \theta_n - \alpha^{\mathrm{cl}}\frac{\mathbf{v}^{\mathrm{cl}}_{n+1}}{\sqrt{\mathbf{w}^{\mathrm{cl}}_{n+1}}+\varepsilon^{\mathrm{cl}}},\\
0\le\beta_1^{\mathrm{cl}}<1,&\quad 0\le\beta_2^{\mathrm{cl}}<1,\quad \alpha^{\mathrm{cl}}>0,\quad \varepsilon^{\mathrm{cl}}>0,
\end{aligned}\tag{29}$$

where squares and square roots of vectors are elementwise.

Adam differs from the methods before it in one property that matters here. Plain gradient descent and momentum are *linear* in the gradients: they add them up, and a sum does not care how its terms were grouped into batches. That is why only the first two moments of an update entered Section 2, and why they took the form (12). An update that divides by a running estimate of the gradient's size is not linear, and it sees the noise of each step directly. Where that leaves the batch size is worked out from Section 5.2 on.

Classic Adam tracks the square of the batch-mean gradient, $\left[b^{-1}\sum g\right]^2$, rather than the mean of the squared gradients, $b^{-1}\sum g^2$. The latter is the natural continuation of the previous sections: like the first moment, it is a plain sum over the batch, so it smooths across samples under exactly the same exponential normalization we used for momentum. So we first take a modified Adam, call it Adam , with that replacement, and ask afterwards what the deviation costs. Adam is a different algorithm with its own hyperparameters, which we mark with a prime to keep them apart from those of Adam itself.

5.1 Adam : replacing the square of the mean

Written in the classic shape, Adam reads

$$\begin{aligned}
\hat{\mathbf{v}}_{n+1} &= \hat{\beta}_1\,\hat{\mathbf{v}}_n + (1-\hat{\beta}_1)\,b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i(\theta_n),\\
\hat{\mathbf{w}}_{n+1} &= \hat{\beta}_2\,\hat{\mathbf{w}}_n + (1-\hat{\beta}_2)\,b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i^2(\theta_n),\\
\theta_{n+1} &= \theta_n - \hat{\alpha}\frac{\hat{\mathbf{v}}_{n+1}}{\sqrt{\hat{\mathbf{w}}_{n+1}}+\hat{\varepsilon}},\\
&0\le\hat{\beta}_1<1,\quad 0\le\hat{\beta}_2<1,\quad \hat{\alpha}>0,\quad \hat{\varepsilon}>0.
\end{aligned}\tag{30}$$

The hats are temporary: this is the parameterization we are about to replace, and these symbols appear only here and in (32).

Our normalization gives Adam the symbols it keeps for the rest of the article:

$$\begin{aligned}
\mathbf{v}'_{n+1} &= e^{-\frac{b}{\tau'_1}}\,\mathbf{v}'_n + (1-e^{-\frac{b}{\tau'_1}})\,b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i(\theta_n),\\
\mathbf{w}'_{n+1} &= e^{-\frac{b}{\tau'_2}}\,\mathbf{w}'_n + (1-e^{-\frac{b}{\tau'_2}})\,b^{-1}\sum_{i:\ t(i)=n}\mathbf{g}_i(\theta_n)^2,\\
\theta_{n+1} &= \theta_n - \alpha'\,b\frac{\mathbf{v}'_{n+1}}{\sqrt{\mathbf{w}'_{n+1}}+\varepsilon'},\\
&0\le\tau'_1,\quad 0\le\tau'_2,\quad \alpha'>0,\quad \varepsilon'>0.
\end{aligned}\tag{31}$$

The time constants τ'_1 and τ'_2 are measured in samples, exactly as τ is in (24).

The two moments are treated differently on purpose. The explicit b in the step does the work that the batch sums do in (7) and (23): the product $b\,\mathbf{v}'$ is a batch sum, so each sample contributes once regardless of how the samples are grouped. The denominator must not be scaled that way, because it sits under a square root: if $\mathbf{w}'$ were a batch sum it would grow like b, its square root like $\sqrt{b}$, and a stray $\sqrt{b}$ would survive in the step. Keeping $\mathbf{w}'$ a per-sample mean makes it a batch-size-independent scale, which is what a normalizer should be.

Setting aside the deviation just introduced, the relations between the two parameterizations of Adam are:

$$\begin{aligned}
\hat{\beta}_1 &= e^{-\frac{b}{\tau'_1}},\\
\hat{\beta}_2 &= e^{-\frac{b}{\tau'_2}},\\
\hat{\alpha} &= b\,\alpha',\\
\hat{\varepsilon} &= \varepsilon'.
\end{aligned}\tag{32}$$

The step size scales with the batch size exactly as it did for plain gradient descent in (6), and ε' does not scale at all. Adam , at least, fits the pattern of the previous sections without a remainder.

5.2 The noise of a batch

Both algorithms are built out of the same raw material, the gradient of a single sample, so we start with that gradient. Take one component of it and read it as a noisy measurement: a signal μ, the same for every sample, plus a fluctuation of typical size σ that differs from sample to sample. In the notation of (11), μ is that component of $\bar{\mathbf{g}}$ and σ^2 is the matching diagonal entry of Σ. Everything that follows is componentwise, and every component carries its own pair.

A batch of b samples adds b such measurements together. Means add, so the signal grows as $b\,\mu$; variances add, so the fluctuation grows only as $\sqrt{b}\,\sigma$. By the central limit theorem, already used in Section 2, the batch sum is moreover close to Gaussian, so its mean and its variance describe it completely. Treating the batch gradient noise this way is standard, as in Mandt, Hoffman and Blei and Li, Tai and E.

The ratio of the two,

$$\mathrm{SNR}(b) = \sqrt{b}\,\frac{\mu}{\sigma}, \tag{33}$$

is the signal to noise ratio of the batch gradient. It passes one at the batch size

$$b_* := \frac{\sigma^2}{\mu^2}, \tag{34}$$

which we call the crossover: below it ($b \ll b_*$) a batch gradient is mostly fluctuation, above it ($b \gg b_*$) mostly signal.

The ratio is not one number for a whole network. It differs between weights, between layers, and between the stages of training. We need only a one-sided statement about it. The argument asks that the crossover lie far above any batch we would run, so the quantity to quote is a bound over the weights and over the training, not a typical value. For the model of this article,

$$\frac{\mu}{\sigma} \lesssim 10^{-2}, \tag{35}$$

with the largest values along the active part of the trajectory and smaller ones on either side of it, which puts the crossover at

$$b_* \gtrsim 10^4. \tag{36}$$

So a single sample carries a very weak signal in this model. Others report the same direction: McCandlish et al. form the same ratio out of sums over the weights, call it the gradient noise scale, and find that it grows as the loss falls over a run. The largest useful batch size they report ranges from the tens of thousands on ImageNet to the millions in a reinforcement learning agent. Theirs is a number of samples, unlike the noise scale of Section 4, which is a temperature. A crossover of 10^4 samples and up sits inside those figures, and on their evidence a larger problem puts it further off.

5.3 Where each algorithm puts the gate

With this in hand we can state the difference between the two algorithms exactly. Over one batch,

$$\mathbb{E}\left[\sum_{\text{batch}} g^2\right] = b\,(\mu^2 + \sigma^2) \tag{37}$$

for the quantity Adam tracks, and

$$\mathbb{E}\left[\Big(\sum_{\text{batch}} g\Big)^2\right] = b^2\mu^2 + b\,\sigma^2 \tag{38}$$

for the one Adam tracks. The first grows as b at every batch size. The second grows as b while $b \ll b_*$ and as b^2 once $b \gg b_*$, which is the crossover again, in the form in which it will be used.

Both algorithms divide by a running average of one of these. The numerators are the same object in both, an exponential moving average of the batch mean gradient, so

$$\mathbb{E}[\mathbf{v}'] = \mathbb{E}\big[\mathbf{v}^{\text{cl}}\big] = \mu. \tag{39}$$

The denominators differ. Adam averages the mean of the squares, the first display divided by b, and Adam the square of the batch mean, the second display divided by b^2, so that

$$\mathbb{E}[\mathbf{w}'] = \mu^2 + \sigma^2 \qquad \text{and} \qquad \mathbb{E}\big[\mathbf{w}^{\text{cl}}\big] = \mu^2 + \frac{\sigma^2}{b}. \tag{40}$$

All four hold once the averages have forgotten their starting values, and on the assumption that μ and σ change little over the span of an average. These are steady values of the averages; the weights go on moving.

The training is decided per sample, by the average weight increment $\mathbb{E}[\Delta\theta]/b$, since one update consumes b samples. Take it for both algorithms at once. Adam moves a weight by $-\alpha' b\,\mathbf{v}'/(\sqrt{\mathbf{w}'} + \varepsilon')$ and Adam by $-\alpha^{\text{cl}}\mathbf{v}^{\text{cl}}/(\sqrt{\mathbf{w}^{\text{cl}}} + \varepsilon^{\text{cl}})$, and in both the ε only keeps the division safe, so drop it. The smoothing is done on $\mathbf{v}$ and $\mathbf{w}$ separately, so we take each increment at their settled values, $\mathbb{E}[\mathbf{v}]$ and $\mathbb{E}[\mathbf{w}]$. What the fluctuations around those values leave behind is of the order of the inverse number of updates an average spans, which is small at the decay rates Adam is usually run with, and the displays below are accurate to that order. The estimate takes the noise of a single sample to be symmetric, and a skewed one enlarges it, the more so the smaller the batch.

With the expectations found above, and taking $\mu > 0$ for the component in hand, since a negative μ only flips the sign of what follows, the step and the increment per sample are

$$\begin{aligned} \mathbb{E}[\Delta\theta] &\approx -\alpha'\,b\,\frac{\mu}{\sqrt{\mu^2 + \sigma^2}} = -b\,\alpha'\,\frac{1}{\sqrt{1 + b_*}}, \\ \frac{\mathbb{E}[\Delta\theta]}{b} &\approx -\alpha'\,\frac{1}{\sqrt{1 + b_*}} \end{aligned} \tag{41}$$

for Adam and

$$\begin{aligned}\mathbb{E}[\Delta\theta] &\approx -\alpha^{\mathrm{cl}}\,\frac{\mu}{\sqrt{\mu^2+\sigma^2/b}} = -\alpha^{\mathrm{cl}}\,\frac{1}{\sqrt{1+b_*/b}},\\ \frac{\mathbb{E}[\Delta\theta]}{b} &\approx -\frac{\alpha^{\mathrm{cl}}}{b}\,\frac{1}{\sqrt{1+b_*/b}} = -\frac{\alpha^{\mathrm{cl}}}{\sqrt{b}\,\sqrt{b+b_*}}\end{aligned} \tag{42}$$

for Adam. Adam takes a step b times the larger, by the explicit factor in (31), and that is exactly the factor which disappears again on dividing by the b samples the step consumed. In Adam nothing cancels the division.

The factor $1/\sqrt{1+\cdot}$ in each is a gate: close to one when the gradient being measured is clean, close to zero when that gradient is mostly noise, and opening as the signal to noise ratio grows. The design is deliberate. Kingma and Ba call the ratio of the two moment estimates a signal to noise ratio, and note that a smaller value should give an effective step closer to zero, precisely because a small value means the direction is uncertain. Balles and Hennig, taking Adam apart, find its direction set by the sign of the gradient and its magnitude by an estimate of the relative variance. The gate is a large part of why Adam is used at all.

The two algorithms place the same gate at different points. Adam measures the signal to noise ratio of a single sample, so its gate sits at b_* and does not move. Adam measures the ratio of the batch gradient it actually steps with, so its gate sits at b_*/b and the batch pushes it open. Where the gate sits is the only difference between the two algorithms.

5.4 What a rescaling can and cannot do

Adam needs no retuning at all, for any component: by Section 5.3 the batch size has already left its increment per sample, and gradient accumulation can buy nothing there.

In Adam's increment it stays, inside $\sqrt{b}\,\sqrt{b+b_*}$. A change of α^{cl} cannot compensate that in general, since b_* varies across weights, layers and training stages, Section 5.2, while α^{cl} is a single number for the whole network.

Two limits are the exception, because in both the batch size separates from b_*:

$$\frac{\mathbb{E}[\Delta\theta]}{b} \approx -\frac{\alpha^{\mathrm{cl}}}{b} \quad \text{for} \quad b \gg b_*, \qquad \frac{\mathbb{E}[\Delta\theta]}{b} \approx -\frac{\alpha^{\mathrm{cl}}}{\sqrt{b}\,\sqrt{b_*}} \quad \text{for} \quad b \ll b_*. \tag{43}$$

The first asks for a linear rescaling, holding α^{cl}/b fixed, the second for a square root one, holding

$$\tilde{\alpha} := \frac{\alpha^{\mathrm{cl}}}{\sqrt{b}} \tag{44}$$

fixed, and each serves every component at once. Between the two, where b and b_* are comparable, no single number serves.

Real training runs at

$$1 \leq b \ll b_*, \tag{45}$$

so the second is the practical one: raising the batch by a factor k asks for the classical step size to be raised by $\sqrt{k}$, which is the square root scaling rule for Adam, see Malladi et al. and Granziol et al.. Malladi et al. prove more than we do: their rule comes from a limit in which the whole training trajectory is preserved, while we have only matched the expected progress per sample. The two exponents agree, and that agreement is not evidence for the stronger claim.

Figure 8 puts the rule to the network of Section 1.2. We run Adam without momentum, $\beta_1^{\mathrm{cl}} = 0$, at the batch sizes $b = 1, 4, 16$, all far below b_*, and let the second moment average over the same number of samples at every batch size,

$$\beta_2^{\mathrm{cl}} = 0.999^{\,b}. \tag{46}$$

Where the training is stable, the curves at a common $\tilde{\alpha}$ lie on top of each other. Held at a common α^{cl} instead, the same runs separate, the larger batch settling to a lower loss.

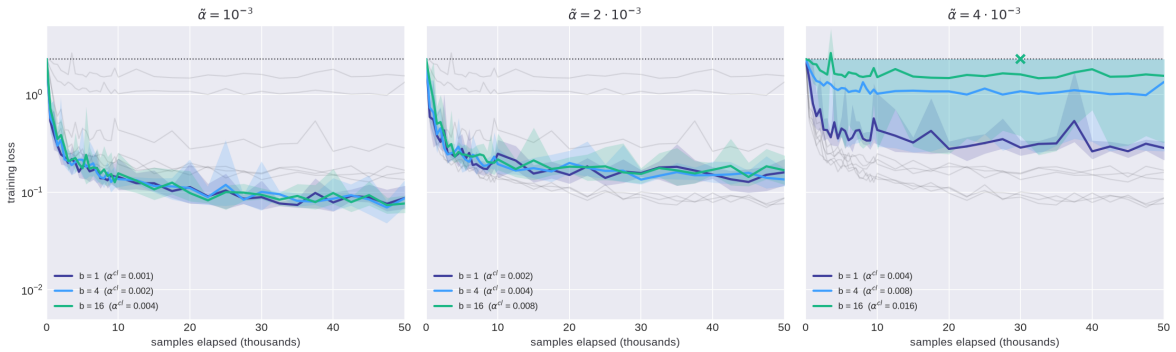

Figure 8: Adam without momentum on MNIST: the training loss against the number of elapsed samples, one panel per $\tilde{\alpha}$. The batch sizes keep their colours from Figure 2, and the legend gives the α^{cl} of each run. Each curve is the geometric mean over five seeds and the band is their spread. A cross marks where a run that stayed at the chance level, the dotted line, was stopped. The grey curves repeat the other panels. At $\tilde{\alpha} = 4 \cdot 10^{-3}$ the training is past its stability boundary, and it is lost in four of five seeds at $b = 16$, three at $b = 4$ and none at $b = 1$.

If moving the gate is worth something, then Adam is the better algorithm and accumulation, which raises the effective batch without the memory, is a legitimate way to move it, at an effective batch of the order of b_* itself, 10^4 samples or more for our model. Nothing in this article establishes that it is worth something, and we do not test whether training there is any better.

5.5 A variant that does scale

Adam's form has a practical argument behind it: $\sum g^2$ requires a separate gradient for every element of the batch, whereas $(\sum g)^2$ comes for free from the batch gradient that back-propagation already produces. That weighs on the choice far more heavily than anything above, and it is most probably why Adam was written with the square of the mean in the first place.

Adam is not an approximation to Adam that happens to be convenient. It is a different algorithm. One could build a variant that does approximate Adam, by replacing the square of the mean with $b^{-2}\sum g^2$ instead of $b^{-1}\sum g^2$, and it would agree with Adam whenever $b\,\mu^2 \ll \sigma^2$. But it would inherit the moving gate along the way, and the clean scaling would be gone. So a variant cannot both track Adam and keep the clean scaling.

Invariance under a change of the batch size does not conflict with using only the batch-summed gradient. Adam's particular combination of the two moments gives up the invariance while keeping the cheap quantities. Consider, for instance,

$$\theta_{n+1} = \theta_n - \alpha''\,\sqrt{b}\,\frac{\mathbf{v}''_{n+1}}{\sqrt{\mathbf{w}''_{n+1} - \left(\mathbf{v}''_{n+1}\right)^2} + \varepsilon''}, \tag{47}$$

where $\mathbf{v}''$ and $\mathbf{w}''$ are moving averages of the batch mean gradient and of its square, the two quantities Adam already keeps, both free of per-example gradients. The two averages share a single time constant, $\tau''_1 = \tau''_2$, which Adam's $\beta_1^{\mathrm{cl}} \neq \beta_2^{\mathrm{cl}}$ do not. Their difference estimates the variance of the batch mean,

$$\mathbb{E}[\mathbf{w}''] - \mathbb{E}[\mathbf{v}'']^2 = \left(\mu^2 + \frac{\sigma^2}{b}\right) - \mu^2 = \frac{\sigma^2}{b}, \tag{48}$$

so that

$$\mathbb{E}[\Delta\theta] \approx -\alpha''\,\sqrt{b}\,\frac{\mu}{\sigma/\sqrt{b}} = -b\,\alpha''\,\frac{\mu}{\sigma}, \qquad \frac{\mathbb{E}[\Delta\theta]}{b} \approx -\alpha''\,\frac{\mu}{\sigma}. \tag{49}$$

The batch size is gone, and the gate is now μ/σ, the signal to noise ratio of a single sample. That is $1/\sqrt{b_*}$, so it agrees with Adam 's $1/\sqrt{1+b_*}$ on every component whose gradient is noisy, which by the measurement above is every component that matters.

Because the two moments share a time constant, the subtraction is a weighted variance of the batch means and cannot be negative; two different time constants would not guarantee that. The gate, however, has no upper bound where Adam's saturates at one, so ε'' is doing real work rather than guarding a division.

Balles and Hennig, cited above for their reading of Adam, build M-SVAG out of the same difference of two averages under a single shared decay, as here, with a correction for the bias it carries. Zhuang et al. reach the same denominator from the other side: AdaBelief averages the squared deviation of the batch mean gradient from its own moving average, which is equally cheap.

We are not proposing this rule and we have not run it.

6 Conclusion

6.1 What This Means in Practice

We would rather not give general recommendations. There are always details that can matter more than anything derived here. What the article offers is an argument about how the hyperparameters ought to behave, and its conclusions agree with the literature in the main.

Read back in the classical variables, it gives each optimizer a rule for keeping the training the same per sample when the batch size is raised by a factor k.

Plain gradient descent, by Section 1, inside the stable region of Section 3:

$$\lambda^{\mathrm{cl}} \to k\,\lambda^{\mathrm{cl}}. \tag{50}$$

Gradient descent with momentum, (17), by Section 4:

$$\begin{aligned} m^{\mathrm{cl}} &\to \left(m^{\mathrm{cl}}\right)^k, \\ \lambda^{\mathrm{cl}} &\to k\,\lambda^{\mathrm{cl}}\,\frac{1-\left(m^{\mathrm{cl}}\right)^k}{1-m^{\mathrm{cl}}}. \end{aligned} \tag{51}$$

The factor approaches k as $m^{\mathrm{cl}} \to 1$, where the rule becomes

$$\lambda^{\mathrm{cl}} \to k^2\,\lambda^{\mathrm{cl}} + O(1-m^{\mathrm{cl}}). \tag{52}$$

The extra factor is the gain of the unnormalized velocity of (17), which changes with m^{cl}. Adam's moments are normalized, so its step size rule carries no such factor.

Adam, (29), by Section 5.4 for the step size and by the same argument as momentum for the decay rates:

$$\begin{aligned} \alpha^{\mathrm{cl}} &\to \sqrt{k}\,\alpha^{\mathrm{cl}}, \\ \beta_i^{\mathrm{cl}} &\to \left(\beta_i^{\mathrm{cl}}\right)^k. \end{aligned} \tag{53}$$

None of these rules is new. The linear one goes back to Krizhevsky and is the rule of Goyal et al., the momentum one holds the noise scale of Smith, Kindermans, Ying and Le fixed and the smoothing length with it, the square root one is that of Malladi et al., and Marek et al. count Adam's decay in tokens.

6.2 What This Does Not Cover

The analytic statements hold to leading order in λ, and the experiments behind them are the single small network on MNIST described at the start.

A small λ is the assumption the whole argument rests on, and near the stability boundary of Section 3 it can stop holding. Training is often pushed as close to that boundary as it will go. Two things change there. The boundary is not a single number: it differs from weight to weight and from layer to layer. And a run that sits on it has no free λ left, because the boundary fixes the value, so λ stops being a hyperparameter in the sense used here.

Two omissions are more than caveats, since they are where the verdict on accumulation could change. Batch normalization makes the loss depend on how the samples are grouped, so an accumulated batch and a real one are not the same object, see Mattias Teye, Hossein Azizpour, Kevin Smith. And losses that are not sums over samples, contrastive objectives being the usual example, fall outside the argument.

6.3 A Word to the Reader

When I first heard of gradient accumulation I was puzzled. That such a thing should be worth doing is open to doubt, and this article is my attempt to say where the doubt comes from. I hope it has reached the reader.

This is not a claim that accumulation is useless. In machine learning anything may work against reasonable expectation. It is a claim about where the burden of proof lies: the usefulness is what needs an argument, and not the doubt. So instead of resting on the doubt, I went looking for the argument that would settle it the other way, and that search is most of this text: it follows the batch size through the drift, the noise, the momentum and Adam, and the one place the batch size survives is inside Adam's gate. Practical examples are not known to me either, and an improvement in the metrics after a head-on increase of the batch size is not one, since the hyperparameters moved along with it.

Counting in samples rather than in updates, or in tokens for a language model, is the more obvious half of all this.

7 Postscript

Some years after the first version of this post appeared, Marek et al. put the same questions to language models, at a scale nothing here approaches. Much of it agrees. They recommend the smallest batch size that saturates throughput, advise against gradient accumulation outside a multi-device case this article does not treat, count the decay of Adam's second moment in tokens rather than in updates, and train stably with no momentum at all, which Section 4 allows.

One thing does not agree. They find that the square root rule of Section 5.4 "does not hold" in their setting: from a batch of one to a batch of 1024 it asks for the step size to be multiplied by 32, and they measure a factor of about 3 as the better choice. Section 5.4 computes the step size that leaves the expected progress per sample unchanged, which is not the same quantity as the step size that trains best. We leave it there.

Revised in 2026 with Claude Code: language, review, checking of the mathematics and the citations, drafting from the author's notes. The results and conclusions are the author's.

References

Lukas Balles, Philipp Hennig, *Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients*, arXiv:1705.07774 (2017).

Alex Krizhevsky, *One Weird Trick for Parallelizing Convolutional Neural Networks*, arXiv:1404.5997 (2014).

Priya Goyal, Piotr Dollár, Ross Girshick et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, arXiv:1706.02677 (2017).

Yarin Gal, Zoubin Ghahramani, *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*, ICML 2016, arXiv:1506.02142.

Diego Granziol, Stefan Zohren, Stephen Roberts, *Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training*, arXiv:2006.09092 (2020).

Stanisław Jastrzębski, Zachary Kenton, Devansh Arpit et al., *Three Factors Influencing Minima in SGD*, arXiv:1711.04623 (2017).

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, Sham M. Kakade, *On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization*, arXiv:1803.05591 (2018).

Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 (2014).

Qianxiao Li, Cheng Tai, Weinan E, *Stochastic modified equations and adaptive stochastic gradient algorithms*, arXiv:1511.06251 (2015).

Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, Sanjeev Arora, *On the SDEs and Scaling Rules for Adaptive Gradient Algorithms*, arXiv:2205.10287 (2022).

Stephan Mandt, Matthew D. Hoffman, David M. Blei, *Stochastic Gradient Descent as Approximate Bayesian Inference*, arXiv:1704.04289 (2017).

Martin Marek, Sanae Lotfi, Aditya Somasundaram et al., *Small Batch Size Training for Language Models: When Vanilla SGD Works, and Why Gradient Accumulation Is Wasteful*, arXiv:2507.07101 (2025).

Sam McCandlish, Jared Kaplan, Dario Amodei, OpenAI Dota Team, *An Empirical Model of Large-Batch Training*, arXiv:1812.06162 (2018).

Gareth O. Roberts, Richard L. Tweedie, *Exponential convergence of Langevin distributions and their discrete approximations*, Bernoulli **2**(4) (1996).

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini et al., *Measuring the Effects of Data Parallelism on Neural Network Training*, arXiv:1811.03600 (2018).

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le, *Don't Decay the Learning Rate, Increase the Batch Size*, arXiv:1711.00489 (2017).

Samuel L. Smith, Quoc V. Le, *A Bayesian Perspective on Generalization and Stochastic Gradient Descent*, arXiv:1710.06451 (2017).

Mattias Teye, Hossein Azizpour, Kevin Smith, *Bayesian Uncertainty Estimation for Batch Normalized Deep Networks*, arXiv:1802.06455 (2018).

Max Welling, Yee Whye Teh, *Bayesian Learning via Stochastic Gradient Langevin Dynamics*, ICML (2011).

Kun Yuan, Bicheng Ying, Ali H. Sayed, *On the Influence of Momentum Acceleration on Online Learning*, Journal of Machine Learning Research **17**(192) (2016).

Juntang Zhuang, Tommy Tang, Yifan Ding et al., *AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients*, arXiv:2010.07468 (2020).