# An Empirical Model of Large-Batch Training

**Sam McCandlish**[*]
OpenAI
sam@openai.com

**Jared Kaplan**
Johns Hopkins University, OpenAI
jaredk@jhu.edu

**Dario Amodei**
OpenAI
damodei@openai.com

and the **OpenAI Dota Team**[†]

## Abstract

In an increasing number of domains it has been demonstrated that deep learning models can be trained using relatively large batch sizes without sacrificing data efficiency. However the limits of this massive data parallelism seem to differ from domain to domain, ranging from batches of tens of thousands in ImageNet to batches of millions in RL agents that play the game Dota 2. To our knowledge there is limited conceptual understanding of why these limits to batch size differ or how we might choose the correct batch size in a new domain. In this paper, we demonstrate that a simple and easy-to-measure statistic called the *gradient noise scale* predicts the largest useful batch size across many domains and applications, including a number of supervised learning datasets (MNIST, SVHN, CIFAR-10, ImageNet, Billion Word), reinforcement learning domains (Atari and Dota), and even generative model training (autoencoders on SVHN). We find that the noise scale increases as the loss decreases over a training run and depends on the model size primarily through improved model performance. Our empirically-motivated theory also describes the tradeoff between compute-efficiency and time-efficiency, and provides a rough model of the benefits of adaptive batch-size training.

[*]Work done as an OpenAI Fellow.

[†]The OpenAI Dota Team (Greg Brockman, Brooke Chan, Przemysław Debiak, Christy Dennison, David Farhi, Rafał Józefowicz, Jakub Pachocki, Michael Petrov, Henrique Pondé, Jonathan Raiman, Szymon Sidor, Jie Tang, Filip Wolski, and Susan Zhang) performed measurements of the reinforcement learning agents they developed for the game Dota 2. The Dota team's work can be cited as [BCD+18].

## Contents

## 1 Introduction

The last few years have seen a rapid increase in the amount of computation used to train deep learning models [AH18]. A major enabler as well as a limiting factor in this growth has been parallelism – the extent to which a training process can be usefully spread across multiple devices. Regardless of how much total computation is available, if model training cannot be sufficiently parallelized, then it may take too much serial time and therefore may be practically infeasible.

A very common source of parallelism in deep learning has been data parallelism, which involves splitting a batch of data across multiple devices and then aggregating and applying the resulting gradients. Data parallelism requires fast communication between devices, but also requires that large batches are algorithmically effective in accelerating learning. Recently, a number of papers have shown empirically that on specific datasets or tasks, large batch sizes can achieve almost linear speed-ups in training without substantially harming sample efficiency or generalization. For example, batch sizes of 8 thousand [GDG+17], 16 thousand [SKYL17], 32 thousand [YGG17, YZH+17, ASF17], and even 64 thousand [JSH+18] examples have been effectively employed to train ImageNet, and batch sizes of thousands have been effective for language models and generative models [OEGA18, PKYC18, BDS18]. This phenomenon is not confined to supervised learning: in reinforcement learning, batch sizes of over a million timesteps (with tens of thousands of environments running in parallel) have been used in a Dota-playing agent [BCD+18], and even in simple Atari environments batch sizes of several thousand timesteps have proved effective [AAG+18, HQB+18, SA18]. These discoveries have allowed massive amounts of data and computation to be productively poured into models in a reasonable amount of time, enabling more powerful models in supervised learning, RL, and other domains.

However, for a given dataset and model, there is little to guide us in predicting how large a batch size we can feasibly use, why that number takes a particular value, or how we would expect it to differ if we used a different dataset or model. For example, why can we apparently use a batch size of over a million when training a Dota agent, but only thousands or tens of thousands when training an image recognition model? In practice researchers tend to simply experiment with batch sizes and see what works, but a downside of this is that large batch sizes often require careful tuning to be effective (for example, they may require a warmup

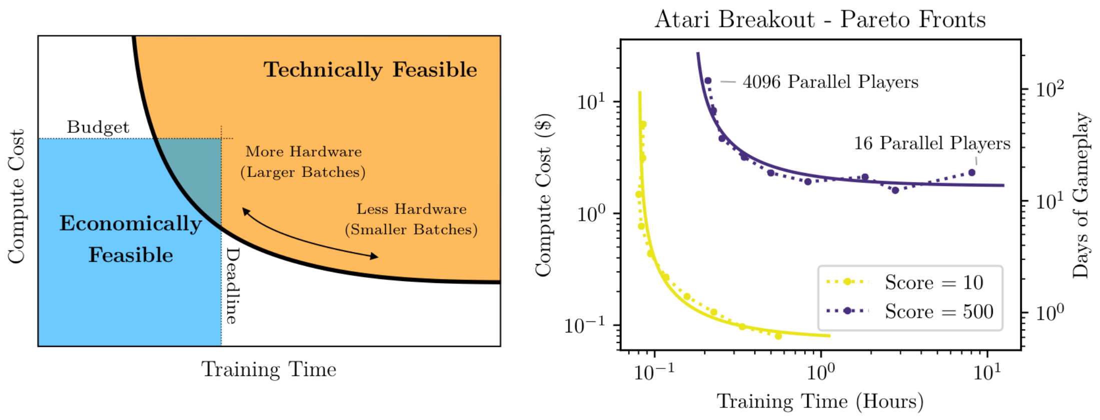


Figure 1: The tradeoff between time and compute resources spent to train a model to a given level of performance takes the form of a Pareto frontier (left). Training time and compute cost are primarily determined by the number of optimization steps and the number of training examples processed, respectively. We can train a model more quickly at the cost of using more compute resources. On the right we show a concrete example of the Pareto frontiers obtained from training a model to solve the Atari Breakout game to different levels of performance. The cost and training time depend on the computing architecture and are shown approximately.

period or an unusual learning rate schedule), so the fact that it is possible to use a large batch size can remain undiscovered for a long time. For example, both the Atari and ImageNet tasks were for several years conventionally run with a substantially smaller batch size than is now understood to be possible. Knowing ahead of time what batch size we expect to be effective would be a significant practical advantage in training new models.

In this paper we attempt to answer some of these questions. We measure a simple empirical statistic, the gradient noise scale[3] (essentially a measure of the signal-to-noise ratio of gradient across training examples), and show that it can approximately predict the largest efficient batch size for a wide range of tasks. Our model also predicts a specific shape for the compute/time tradeoff curve, illustrated in Figure 1. Our contributions are a mix of fairly elementary theory and extensive empirical testing of that theory.

On the conceptual side, we derive a framework which predicts, under some basic assumptions, that training should parallelize almost linearly up to a batch size equal to the noise scale, after which there should be a smooth but relatively rapid switch to a regime where further parallelism provides minimal benefits. Additionally, we expect that the noise scale should increase during training as models get more accurate, and should be larger for more complex tasks, but should not have a strong dependence on model size per se. We also provide an analysis of the efficiency gains to be expected from dynamically adjusting the batch size according to noise scale during training. Finally, we predict that, all else equal, the noise scale will tend to be larger in complex RL tasks due to the stochasticity of the environment and the additional variance introduced by the credit assignment problem.

On the empirical side, we verify these predictions across 8 tasks in supervised learning, RL, and generative models, including ImageNet, CIFAR-10, SVHN, MNIST, BillionWord, Atari, OpenAI's Dota agent [BCD+18], and a variational autoencoder for images. For each of these tasks we demonstrate that the noise scale accurately predicts the largest usable batch size (at the order of magnitude level) and that gains to parallelism degrade in the manner predicted by theory. We also show that the noise scale increases over the course of training and demonstrate efficiency gains from dynamic batch size tuning. The noise scale eventually becomes larger for more performant models, but this appears to be caused by the fact that more performant models simply achieve a better loss.

The rest of this paper is organized as follows. In Section 2, we derive a simple conceptual picture of the noise scale, data parallelism, and batch sizes, and explain what it predicts about optimal batch sizes and how

[3]Similar metrics have appeared previously in the literature. We discuss related work in Section 4.

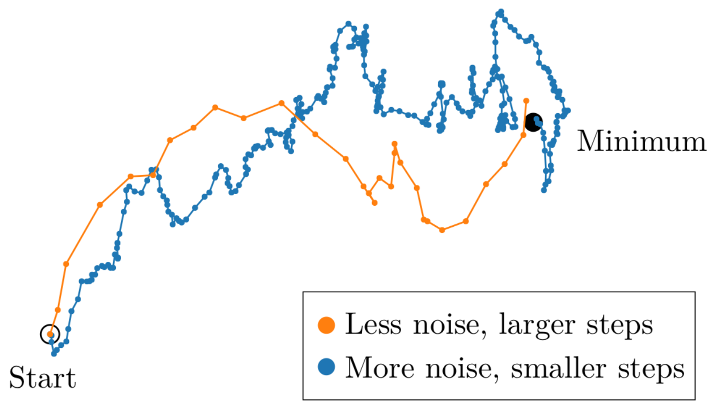


Figure 2: Less noisy gradient estimates allow SGD-type optimizers to take larger steps, leading to convergence in a smaller number of iterations. As an illustration, we show two optimization trajectories using momentum in a quadratic loss, with different step sizes and different amounts of artificial noise added to the gradient.

they vary over the course of training and across tasks. We build on this analysis to study training efficiency in Section 2.3. Then in Section 3 we empirically test the predictions in Section 2 and explore how the noise scale varies with dataset, model size, and learning paradigm (supervised learning vs RL vs generative models). Section 4 describes related work and Section 5 discusses the implications of these results and possible future experiments.

## 2 Theory and Predictions for the Gradient Noise Scale

### 2.1 Intuitive Picture

Before working through the details of the gradient noise scale and the batch size, it is useful to present the intuitive picture. Suppose we have a function we wish to optimize via stochastic gradient descent (SGD). There is some underlying true optimization landscape, corresponding to the loss over the entire dataset (or, more abstractly, the loss over the distribution it is drawn from). When we perform an SGD update with a finite batch size, we're approximating the gradient to this true loss. How should we decide what batch size to use?

When the batch size is very small, the approximation will have very high variance, and the resulting gradient update will be mostly noise. Applying a bunch of these SGD updates successively will average out the variance and push us overall in the right direction, but the individual updates to the parameters won't be very helpful, and we could have done almost as well by aggregating these updates in parallel and applying them all at once (in other words, by using a larger batch size). For an illustrative comparison between large and small batch training, see Figure 2.

By contrast, when the batch size is very large, the batch gradient will almost exactly match the true gradient, and correspondingly two randomly sampled batches will have almost the same gradient. As a result, doubling the batch size will barely improve the update – we will use twice as much computation for little gain.

Intuitively, the transition between the first regime (where increasing the batch size leads to almost perfectly linear speedups) and the second regime (where increasing the batch size mostly wastes computation) should occur roughly where the noise and signal of the gradient are balanced – where the variance of the gradient is at the same scale as the gradient itself[4]. Formalizing this heuristic observation leads to the noise scale.

The situation is shown pictorially in Figure 1. For a given model, we'd like to train it in as little wall time as possible (x-axis) while also using as little total computation as possible (y-axis) – this is the usual goal

[4]Note that these considerations are completely agnostic about the size of the dataset itself.

of parallelization. Changing the batch size moves us along a tradeoff curve between the two. Initially, we can increase the batch size without much increase in total computation, then there is a "turning point" where there is a substantive tradeoff between the two, and finally when the batch size is large we cannot make further gains in training time. In the conceptual and experimental results below, we formalize these concepts and show that the bend in the curve (and thus the approximate largest effective batch size) is in fact set roughly by the noise scale.

### 2.2 Gradients, Batches, and the Gradient Noise Scale

We'll now formalize the intuitions described in Section 2.1. Consider a model, parameterized by variables $\theta \in \mathbb{R}^D$, whose performance is assessed by a loss function $L(\theta)$. The loss function is given by an average over a distribution $\rho(x)$ over data points $x$. Each data point $x$ has an associated loss function $L_x(\theta)$, and the full loss is given by $L(\theta) = \mathbb{E}_{x\sim\rho}[L_x(\theta)]$[5].

We would like to minimize $L(\theta)$ using an SGD-like optimizer, so the relevant quantity is the gradient $G(\theta) = \nabla L(\theta)$. However, optimizing $L(\theta)$ directly would be wasteful if not impossible, since it would require processing the entire data distribution every optimization step. Instead, we obtain an estimate of the gradient by averaging over a collection of samples from $\rho$, called a batch:

$$G_{\text{est}}(\theta) = \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta L_{x_i}(\theta); \qquad x_i \sim \rho \tag{2.1}$$

This approximation forms the basis for stochastic optimization methods such as mini-batch stochastic gradient descent (SGD) and Adam [KB14]. The gradient is now a random variable whose expected value (averaged over random batches) is given by the true gradient. Its variance scales inversely with the batch size $B$[6]:

$$\begin{aligned}\mathbb{E}_{x_{1\cdots B}\sim\rho}[G_{\text{est}}(\theta)] &= G(\theta)\\ \text{cov}_{x_{1\cdots B}\sim\rho}(G_{\text{est}}(\theta)) &= \frac{1}{B}\Sigma(\theta),\end{aligned} \tag{2.2}$$

where the per-example covariance matrix is defined by

$$\begin{aligned}\Sigma(\theta) &\equiv \text{cov}_{x\sim\rho}(\nabla_\theta L_x(\theta))\\ &= \mathbb{E}_{x\sim\rho}\left[(\nabla_\theta L_x(\theta))(\nabla_\theta L_x(\theta))^T\right] - G(\theta)G(\theta)^T.\end{aligned} \tag{2.3}$$

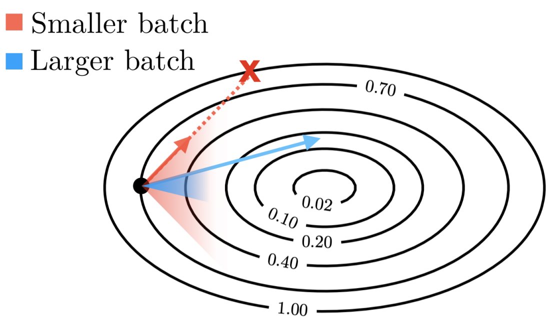
The key point here is that the minibatch gradient gives a noisy estimate of the true gradient, and that larger batches give higher quality estimates. We are interested in how useful the gradient is for optimization purposes as a function of $B$, and how that might guide us in choosing a good $B$. We can do this by connecting the noise in the gradient to the maximum improvement in true loss that we can expect from a single gradient update. To start, let $G$ denote the true gradient and $H$ the true Hessian at parameter values $\theta$. If we perturb the parameters $\theta$ by some vector $V$ to $\theta - \epsilon V$, where $\epsilon$ is the step size, we can expand true loss at this new point to quadratic order in $\epsilon$:

$$L(\theta - \epsilon V) \approx L(\theta) - \epsilon G^T V + \frac{1}{2}\epsilon^2 V^T H V. \tag{2.4}$$

If we had access to the noiseless true gradient $G$ and used it to perturb the parameters, then Equation 2.4 with $V = G$ would be minimized by setting $\epsilon = \epsilon_{\max} \equiv \frac{|G|^2}{G^T H G}$. However, in reality we have access only to the noisy estimated gradient $G_{\text{est}}$ from a batch of size $B$, thus the best we can do is minimize the expectation

[5]In the context of reinforcement learning, the loss could be the surrogate policy gradient loss, and the distribution $\rho$ would be nonstationary.

[6]This is strictly true only when training examples are sampled independently from the same data distribution. For example, when batches are sampled without replacement from a dataset of size $D$, the variance instead scales like $\left(\frac{1}{B} - \frac{1}{D}\right)$. For simplicity, we restrict ourself to the case where $B \ll D$ or where batches are sampled *with* replacement, but our conclusions can be altered straightforwardly to account for correlated samples.



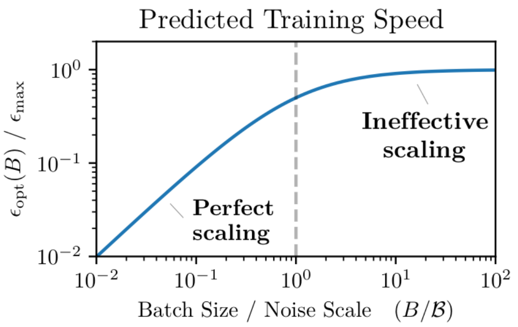


Figure 3: Larger batch sizes yield estimated gradients that are closer to the true gradient, on average. Larger step sizes can be used when the estimated gradient is closer to the true gradient, so more progress can be made per step. **Left:** A large step size used with a small batch size can lead to instability, as illustrated for a quadratic loss. **Right:** Equation 2.6 predicts that the 'turning point' after which larger batch sizes become less helpful is the noise scale $\mathcal{B}$, where the training speed drops to 50% of the maximum possible.

$\mathbb{E}[L(\theta - \epsilon G_{\text{est}})]$ with respect to $\epsilon$. This expected value can be evaluated using Equation 2.2:

$$\mathbb{E}[L(\theta - \epsilon G_{\text{est}})] = L(\theta) - \epsilon|G|^2 + \frac{1}{2}\epsilon^2 \left( G^T H G + \frac{\text{tr}(H\Sigma)}{B} \right). \tag{2.5}$$

Minimizing this equation with respect to $\epsilon$ leads to:

$$\epsilon_{\text{opt}}(B) = \text{argmin}_{\epsilon} \mathbb{E}[L(\theta - \epsilon G_{\text{est}})] = \frac{\epsilon_{\max}}{1 + \mathcal{B}_{\text{noise}}/B} \tag{2.6}$$

as the optimal step size, which produces an optimal improvement in the loss from the noisy gradient:

$$\Delta L_{\text{opt}}(B) = \frac{\Delta L_{\max}}{1 + \mathcal{B}_{\text{noise}}/B}; \qquad \Delta L_{\max} = \frac{1}{2}\frac{|G|^4}{G^T H G}. \tag{2.7}$$

Above, we have defined the *noise scale* as:

$$\mathcal{B}_{\text{noise}} = \frac{\text{tr}(H\Sigma)}{G^T H G}, \tag{2.8}$$

Note that our definition of the noise scale is independent of the size of the full training set. If we use a step size larger than twice $\epsilon_{\text{opt}}$, the loss may *increase*, leading to divergence, as illustrated in Figure 3.

Despite the many unfounded assumptions in the above derivation, we will find that equations 2.7 and 2.8 provide a helpful guide to the behavior of large-batch training, even when using other optimizers (including momentum, Adam, and RMSProp).

For a discussion of the dependence of the noise scale on the learning rate, see Appendix C on the 'temperature' of training.

## Implications and Simplifications

Equation 2.7 implies that when the batch size is much smaller than the noise scale, $B \ll \mathcal{B}_{\text{noise}}$, the second term in the denominator dominates the first, so increasing the batch size $B$ linearly increases the progress in loss. This is the small batch regime, where increases in batch size linearly speed up training. By contrast, when $B \gg \mathcal{B}_{\text{noise}}$, then the first term dominates, so that increasing $B$ has almost no effect on the progress in loss. This is the large batch regime where increases in batch size do not speed up training and simply waste computation; the switch between the two occurs at $B \approx \mathcal{B}_{\text{noise}}$ (see Figure 3).

The noise scale in Equation 2.8 requires some overhead to compute due to the presence of the Hessian $H$. We can estimate it by measuring $\Delta L_{\text{opt}}(B)$ using a series of line searches in the direction of a gradient measured with various batch sizes $B$ and fitting the result to Equation 2.7. This allows us to estimate $\mathcal{B}_{\text{noise}}$ as well as to empirically test whether Equation 2.7 actually fits the data (we discuss these local tests more in Section 3).

The situation gets even simpler if we make the (unrealistic) assumption that the optimization is perfectly well-conditioned – that the Hessian is a multiple of the identity matrix. If that is the case, then Equation 2.8 reduces to:

$$\mathcal{B}_{\text{simple}} = \frac{\operatorname{tr}(\Sigma)}{|G|^2}, \tag{2.9}$$

which says that the noise scale is equal to the sum of the variances of the individual gradient components, divided by the global norm of the gradient[7] – essentially a measure of how large the gradient is compared to its variance. It is also a measure of the scale at which the estimated and true gradient become close in $L^2$ space (having non-trivial dot product) – the expected normalized $L^2$ distance is given by:

$$\frac{\mathbb{E}\left[|G_{\text{est}} - G|^2\right]}{|G|^2} = \frac{1}{B}\frac{\operatorname{tr}(\Sigma)}{|G|^2} = \frac{\mathcal{B}_{\text{simple}}}{B}, \tag{2.10}$$

In practice, we find that $\mathcal{B}_{\text{simple}}$ and $\mathcal{B}_{\text{noise}}$ typically differ only by a small constant multiplicative factor, particularly when we employ common training schemes that improve conditioning. In our empirical work we will sometimes compute $\mathcal{B}_{\text{noise}}$, but will primarily compute $\mathcal{B}_{\text{simple}}$ instead, as it requires less computational expense. In Appendix A.1, we provide an extremely simple method to measure this simplified noise scale with negligible overhead in the context of data-parallel training.

## 2.3 Predictions for Data/Time Efficiency Tradeoffs

Thus far our analysis has only involved a single point in the loss landscape. But in Section 3 we will show that Equation 2.7 nevertheless predicts the dependence of training speed on batch size remarkably well, even for full training runs that range over many points in the loss landscape. By averaging Equation 2.7 over multiple optimization steps (see Appendix D), we find a simple relationship between training speed and data efficiency:

$$\frac{S}{S_{\text{min}}} - 1 = \left(\frac{E}{E_{\text{min}}} - 1\right)^{-1}. \tag{2.11}$$

Here, $S$ and $S_{\text{min}}$ represent the actual and minimum possible number of steps taken to reach a specified level of performance, respectively, and $E$ and $E_{\text{min}}$ represent the actual and minimum possible number of training examples processed to reach that same level of performance. Since we are training at fixed batch size[8], we have $E_{\text{tot}} = BS_{\text{tot}}$. We define the *critical batch size* by an empirical fit to the above equation, as

$$\mathcal{B}_{\text{crit}} = \frac{E_{\text{min}}}{S_{\text{min}}}. \tag{2.12}$$

Our model predicts $\mathcal{B}_{\text{crit}} \approx \mathcal{B}_{\text{noise}}$, where $\mathcal{B}_{\text{noise}}$ is appropriately averaged over training (see Appendix D). Note that the noise scale can vary significantly over the course of a training run, so the critical batch size also depends on the level of performance to which we train the model.

The resulting tradeoff curve in serial time vs total compute has a hyperbolic shape represented in Figure 1. The goal of optimization is to reach a given level of performance with minimal $S$ and $E$ – but as depicted in Figure 1, there are tradeoffs involved, as very small $S$ may require very large $E$, and vice versa. When we choose $B = \mathcal{B}_{\text{crit}}$, the two sides of Equation 2.11 are both 1, so that training takes twice as many passes through the training data as an optimally data-efficient (small-batch) run would take, and twice as many optimization steps as an optimally time-efficient (large-batch) run would take.

[7]One might also use preconditioned gradients, obtained for example by dividing gradient components by the square root of the Adam optimizer's [KB14] accumulated variances. We experimented with this but found mixed results.

[8]We discuss the benefits of dynamically varying the batch size in Appendix D

### 2.4 Assumptions and Caveats

The mathematical argument in the previous sections depends on several assumptions and caveats, and it is useful to list these all in one place, in order to help clarify where and why we might expect the quantities in equations 2.8 and 2.9 to be relevant to training:

1. **Short-horizon bias:** The picture in Section 2.2 is a strictly local picture – it tells us how to best improve the loss on the next gradient step. Greedily choosing the best local improvement is generally not the best way to globally optimize the loss (see e.g. [WRLG18]). For example, greedy optimization might perform poorly in the presence of bad local minima or when the landscape is ill-conditioned. The critical batch size would then be reduced by the extent to which noise is beneficial.
2. **Poor conditioning:** In poorly conditioned optimization problems, parameter values often oscillate along the large-curvature directions rather than decreasing in a predictable way (see e.g. [Goh17] and Appendix E.1). This means that Equation 2.7 will not perfectly reflect the amount of optimization progress made per step. Nevertheless, we will see that it still accurately predicts the *relative* speed of training at different batch sizes via the resulting tradeoff Equation 2.11.
3. **Simplified noise scale:** As noted in Section 2.2, whenever we use the simplified noise scale (Equation 2.9) rather than the exact noise scale (Equation 2.8), this number may be inaccurate to the extent that the Hessian is not well-conditioned. Different components of the gradient can have very different noise scales.
4. **Learning rate tuning:** The arguments in Section 2.2 assume that we take the optimal step size and maximize the expected improvement in loss, Equation 2.6. In practice learning rates are unlikely to be perfectly tuned, so that the actual improvement in loss (and thus the scaling of training with batch size) may not perfectly reflect Equation 2.7. However, by trying to choose the best learning rate schedules (or by simply doing a grid search) we can reduce this source of error. In addition, the noise scale depends strongly on the learning rate via a 'temperature' of training, though this source of error is small as long as the learning rate is reasonably close to optimal. We provide a more detailed discussion of this dependence in Appendix C.
5. **Quadratic approximation:** The Taylor expansion in Equation 2.4 is only to second order, so if third order terms are important, in either the distribution of gradient samples or the optimization landscape, then this may introduce deviations from our conceptual model, and in particular deviations from Equation 2.7. Intuitively, since parameter updates are local and often quite small we suspect that the previous two sources of error will be more important than this third one.
6. **Generalization:** The picture in Section 2.2 says nothing about generalization – it is strictly about optimizing the training loss as a mathematical function. Some papers have reported a "generalization gap" in which large batch sizes lead to good training loss but cause a degradation in test loss, apparently unrelated to overfitting [KMN$^+$16, HHS17]. The arguments in Section 2.2 don't exclude this possibility, but recent work [SLA$^+$18] has found no evidence of a generalization gap when hyperparameters are properly tuned.

Despite these potential issues in our conceptual model, we'll show in Section 3 that the noise scale is overall a good empirical predictor of the critical batch size. Furthermore, we will see that most training runs fit Equation 2.11 remarkably well.

### 2.5 Expected Patterns in the Noise Scale

In the next section we will measure the noise scale for a number of datasets and confirm its properties. However, it is worth laying out a few patterns we would expect it to exhibit on general grounds:

- **Larger for difficult tasks:** We expect $\mathcal{B}$ to be larger for more complex/difficult[9] tasks, because individual data points will be less correlated, or only correlated in a more abstract way. This may

[9]To be clear, we do not expect this to be the primary difference between more and less difficult tasks. Other difficulty metrics such as the intrinsic dimensionality [LFLY18] appear to be unrelated to the amount of gradient noise, though it would be interesting if there were some connection.

apply both over the course of training on a given dataset (where we may pick the 'low-hanging fruit' first, leaving a long tail of more complicated things to learn) or in moving from easier to harder datasets and environments. In reinforcement learning, we expect environments with sparse rewards or long time horizons to have larger noise scale. We also expect generative models to have smaller $\mathcal{B}$ as compared to classifiers training on the same dataset, as generative models may obtain more information from each example.

- **Growth over training:** $\mathcal{B}$ will grow when the gradient decreases in magnitude, as long as the noise $\operatorname{tr}(\Sigma)$ stays roughly constant. Since $|G|$ decreases as we approach the minimum of a smooth loss, we would expect $\mathcal{B}$ to increase during neural network training.
- **Weak dependence on model size:** The number of model parameters in the neural network cancels in the noise scale, so we do not expect $\mathcal{B}$ to exhibit a strong dependence on model size (at fixed loss). As discussed above, models that achieve better loss will tend to have a higher noise scale, and larger models often achieve better loss, so in practice we do expect larger models to have higher noise scale, but only through the mechanism of achieving better loss.
- **Learning rate tuning:** The noise scale will be artificially inflated if the learning rate is too small, due to the 'temperature' dependence described in Appendix C. To get a useful measurement of the noise scale, the learning rate needs to be appropriate to the current point in parameter space.

The first and last points can be exhibited analytically in toy models (see Appendix C), but we do not expect theoretical analyses to provide a great deal of insight beyond the intuitions above. Instead, we will focus on confirming these expectations empirically.

### 2.6 Summary

To summarize, our model makes the following predictions about large-batch training:

- The tradeoff between the speed and efficiency of neural network training is controlled by the batch size and follows the form of Equation 2.11.
- The critical batch size $\mathcal{B}_{\text{crit}}$ characterizing cost/time tradeoffs can be predicted at the order of magnitude level by measuring the gradient noise scale, most easily in the simplified form $\mathcal{B}_{\text{simple}}$ from Equation 2.9.
- The noise scale can vary significantly over the course of a training run, which suggests that the critical batch size also depends on the chosen level of model performance.
- The noise scale depends on the learning rate via the 'temperature' of training, but is consistent between well-tuned training runs (see Appendix C).

## 3 Experiments

We now test the predictions of Section 2 on a range of tasks, including image classification, language modeling, reinforcement learning, and generative modeling. The tasks range from very simple (MNIST) to very complex (5v5 Dota), which allows us to test our model's predictions in drastically varying circumstances. Our central experimental test is to compare the prediction made by the gradient noise scale $\mathcal{B}_{\text{simple}}$ for each task, to the actual limits of batch size $\mathcal{B}_{\text{crit}}$ found by carefully tuned full training runs at an exhaustive range of batch sizes. The overall results of this comparison are summarized in Figure 4. We find that the gradient noise scale predicts the critical batch size at the order of magnitude level, even as the latter varies from 20 (for an SVHN autoencoder) to over 10 million (consistent with prior results reported in [BCD+18]). Details about the hyperparameters, network architectures, and batch size searches are described in Appendix A.4. Below we describe the individual tasks, the detailed measurements we perform on each task, and the results of these measurements.

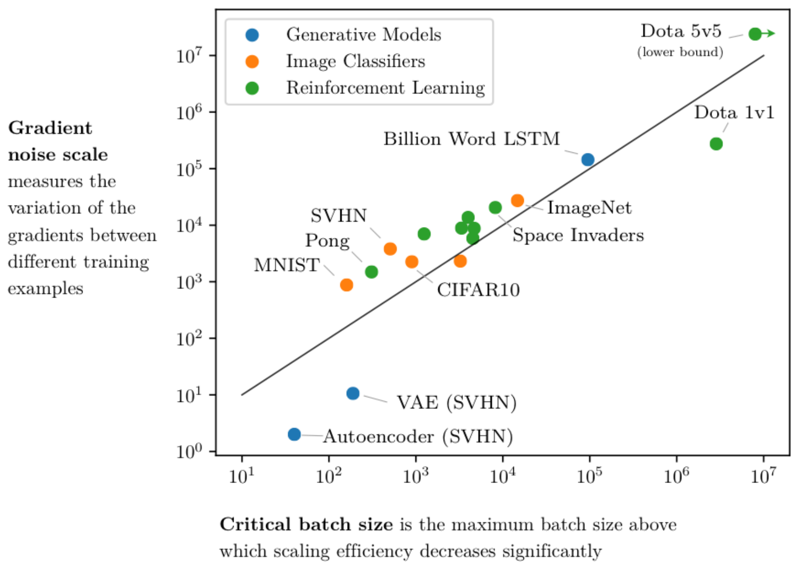


Figure 4: The "simple noise scale" roughly predicts the maximum useful batch size for many ML tasks. We define this "critical batch size" to be the point at which compute efficiency drops below 50% optimal, at which point training speed is also typically 50% of optimal. Batch sizes reported in number of images, tokens (for language models), or observations (for games). We show the critical batch size for a full training run, and the noise scale appropriately averaged over a training run (see Appendix D). Due to resource constraints, for Dota 5v5 we show the batch size used by the OpenAI Dota team as a lower bound for the critical batch size.

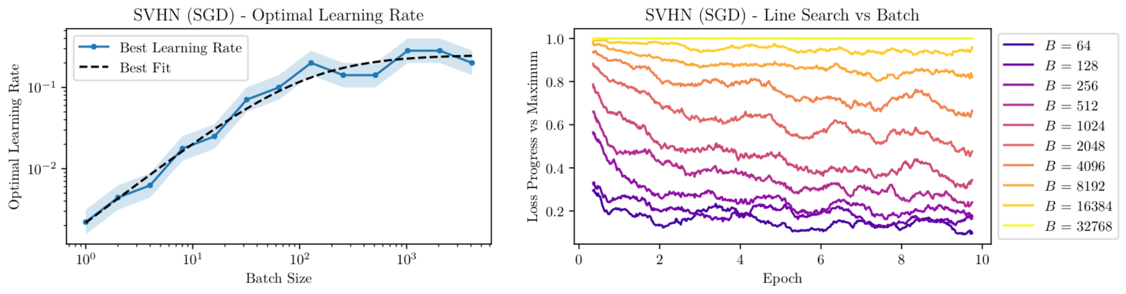


Figure 5: **Left:** The optimal learning rate is displayed for a range of batch sizes, for an SVHN classifier trained with SGD. The optimal learning rate initially scales linearly as we increase the batch size, leveling off in the way predicted by Equation 2.7. **Right:** For a range of batch sizes, we display the average loss progress $\Delta L(B)$ that can be made from a batch of size $B$ via a line search, normalized by the measured $\Delta L(B_{\max})$. Early in training, smaller batches are sufficient to make optimal progress, while larger batches are required later in training.

## 3.1 Quantities Measured

In order to test our model we train each task on a range of batch sizes, selecting the optimal constant learning rate separately for each batch size using a simple grid search. Across a range of tasks, we produce the following results and compare with our model:

- **Optimal learning rates:** When optimizing with plain SGD or momentum, we find that the optimal learning rate follows the functional form of Equation 2.6, as shown in Figure 5. For Adam and RMSProp the optimal learning rate initially obeys a power law $\epsilon(B) \propto B^{\alpha}$ with $\alpha$ between 0.5 and 1.0 depending on the task, then becomes roughly constant. The scale at which the optimal learning

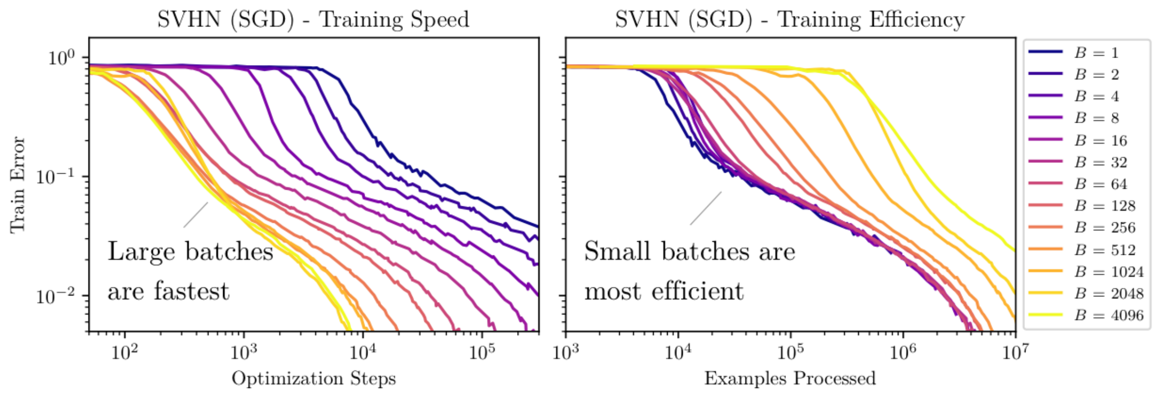


Figure 6: Training runs for a simple CNN classifier on the SVHN dataset at constant batch sizes. Small batch training is more compute-efficient (right), while large-batch training requires fewer optimizer steps (left). The turning point between time-efficient and compute-efficient training occurs roughly at $B = 64$ for the initial phase of training and increases later in training.

rate stops increasing is generally somewhat smaller than the typical noise scale. (See Appendix E.2 for a potential explanation for this power law behavior.)

- **Pareto frontiers:** For each batch size, we observe the number of optimization steps and total number of data samples needed to achieve various levels of performance. This allows us to visualize the tradeoff between time-efficiency and compute-efficiency as a Pareto frontier (see Figures 6 and 7). We find that Equation 2.11 fits the shape of these tradeoff curves remarkably well in most cases.
- **Critical batch size** ($\mathcal{B}_{\text{crit}}$): We determine the critical batch size over the course of a training run by fitting the Pareto fronts to the functional form of Equation 2.11 (see Figure 7). This quantifies the point at which scaling efficiency begins to drop. In particular, training runs at batch sizes much less than $\mathcal{B}_{\text{crit}}$ behave similarly per training example, while training runs at batch sizes much larger than $\mathcal{B}_{\text{crit}}$ behave similarly per optimization step (see Figure 6). The critical batch size typically increases by an order of magnitude or more over the course of training.
- **Simple noise scale** ($\mathcal{B}_{\text{simple}}$): We measure the simple noise scale of Equation 2.9 over the course of a single training run using the minimal-overhead procedure described in Appendix A.1. Note that some care must be taken to obtain a proper estimate of $\mathcal{B}_{\text{simple}}$ due to its dependence on the learning rate via the 'temperature' of training. We find that the noise scale agrees between different well-tuned training runs when compared at equal values of the loss, so it can be accurately measured at small batch size (see Appendix C). We also find that, aside from some fluctuations early in training, $\mathcal{B}_{\text{simple}}$ typically predicts the critical batch size at the order of magnitude level (see Figure 7). The noise scale also typically increases over the course of training, tracking the critical batch size. To obtain a single value for the noise scale representing a full training run, we average over a training run as described in Appendix D.
- **Full noise scale** ($\mathcal{B}_{\text{noise}}$): For SVHN trained with SGD, we also measure the full noise scale $\mathcal{B}_{\text{noise}}$ by performing line searches for gradients obtained by batches of varying size, then fit to the functional form 2.11 (see Figures 5 and 7). This is a somewhat better estimate of $\mathcal{B}_{\text{crit}}$ but is less computationally convenient, so we choose to focus on $\mathcal{B}_{\text{simple}}$ for the remaining tasks.

## 3.2 Results

We summarize our findings in Figure 4: across many tasks, the typical simple noise scale approximately predicts the batch size at which the returns from increasing scale begin to diminish significantly. Results for all of the tasks can be found in Appendix B. We provide a detailed discussion of our methods in Appendix A.

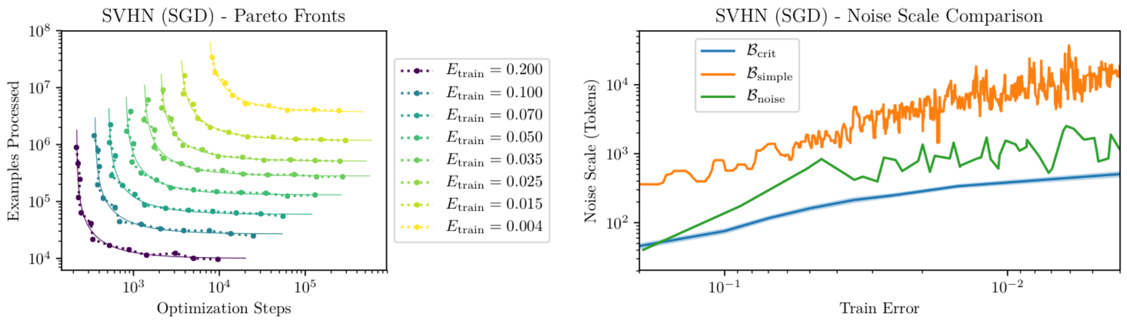


Figure 7: The tradeoff between time-efficiency and compute-efficiency can be visualized as a Pareto frontier. Each point on the diagram above (left) represents the number of optimizer steps and processed examples needed to achieve a particular level of classification accuracy. Fits to Equation 2.11 are also shown.

### Supervised Learning

Basic image classification results are pictured in Figure 14.

- **SVHN** We train a simple CNN image classifier on the extended SVHN dataset [NWC+11]. We display all three of $\mathcal{B}_{\rm crit}$, $\mathcal{B}_{\rm simple}$, and $\mathcal{B}_{\rm noise}$ for SVHN optimized with SGD in Figure 7. We find that $\mathcal{B}_{\rm noise}$ better-predicts $\mathcal{B}_{\rm crit}$ as compared to the more naive $\mathcal{B}_{\rm simple}$. We compare to training using the Adam optimizer [KB14] in Figure 14, where $\mathcal{B}_{\rm simple}$ provides a very accurate prediction for $\mathcal{B}_{\rm crit}$.
- **MNIST** We train a simple CNN on the MNIST dataset [LC10] using SGD, and find that $\mathcal{B}_{\rm simple}$ roughly estimates $\mathcal{B}_{\rm crit}$, though the latter is significantly smaller.
- **CIFAR10** We train a size 32 ResNet [HZRS15] with Momentum on CIFAR10 [Kri09] and find that $\mathcal{B}_{\rm simple}$ predicts $\mathcal{B}_{\rm crit}$.
- **ImageNet** We train a size 50 ResNet [HZRS15] with Momentum on ImageNet [DDS+09], and use a learning rate schedule that decays the learning rate three times during training. Due to the schedule, both $\mathcal{B}_{\rm simple}$, and $\mathcal{B}_{\rm crit}$ change significantly during training (see Appendix C for a discussion) and must be measured separately at each learning rate. Results are pictured in Figure 10. We find that the noise scale varies from 2,000 to 100,000 in the main phase of training, which matches empirical work (e.g. [JSH+18]) showing that constant batch sizes up to 64 thousand can be used without a loss of efficiency. During the later fine-tuning phase of training, the noise scale increases further to hundreds of thousands and even millions, suggesting that even larger batch sizes may be useful at the very end of training. Our critical batch sizes are slightly lower (15k vs 64k) than those reported in the literature, but we did not use the latest techniques such as layer-wise adaptive learning rates [YGG17].

Overall we find that more complex image datasets have larger noise scales in a way that is not directly determined by dataset size.

### Generative Modeling

The results for these tasks are pictured in Figure 9.

- **VAE and Autoencoder** We train a VAE [KW13] and a simple Autoencoder on the SVHN dataset [NWC+11]; we were motivated to compare these models because VAEs introduce additional stochasticity. As expected, the VAE had larger $\mathcal{B}_{\rm crit}$ and $\mathcal{B}_{\rm simple}$ as compared to the Autoencoder, and both models had much lower $\mathcal{B}_{\rm simple}$ as compared to SVHN image classifiers. However, unlike most of the other tasks, for these generative models $\mathcal{B}_{\rm simple}$ was significantly *smaller* than $\mathcal{B}_{\rm crit}$.
- **Language Modeling** We train a single-layer LSTM for autoregressive prediction on the Billion Word dataset [CMS+13], and find good agreement between $\mathcal{B}_{\rm crit}$ and $\mathcal{B}_{\rm simple}$. We also illustrate the

dependence on LSTM size in Figure 8, finding that the noise scale is roughly independent of LSTM size at fixed values of the loss, but that larger LSTMs eventually achieve lower values of the loss and a larger noise scale.

**Reinforcement Learning**

- **Atari** We train RL agents with the policy gradient algorithm A2C [MBM$^+$16] on seven Atari games [BNVB12] (Alien, Beamrider, Breakout, Pong, Qbert, Seaquest, Space Invaders), with results pictured in Figures 11 and 12. The tradeoff curves generally agree well with the prediction of Equation 2.11, though they are somewhat noisy e.g. for Pong since we do not average over multiple seeds. For some Atari games, we find some consistent deviation from 12 at very small batch sizes (see e.g. Beam Rider in Figure 11). It would be interesting to study this phenomenon further, though this could simply indicate greater sensitivity to other hyperparameters (e.g. momentum) at small batch size. Overall, we see that patterns in the noise scale match intuition, such as Pong being much easier to learn than other Atari games.
- **Dota** The OpenAI Dota team has made it possible to train PPO [SWD$^+$17] agents on both Dota 1v1 and 5v5 environments (the latter being preliminary ongoing work). We vary the two hyperparameters batch size and learning rate on the existing code, experiment setup, and training infrastructure as described in [BCD$^+$18]. The Dota 1v1 environment features two agents fighting in a restricted part of the map (although they are free to walk anywhere) with a fixed set of abilities and skills, whereas Dota 5v5 involves the whole map, 5 heroes on each side, and vastly more configurations in which heroes might engage each other. This is reflected in the higher noise scale for Dota 5v5 (at least 10 million) relative to Dota 1v1 – we suspect the higher diversity of situations gives rise to more variance in the gradients. Due to resource constraints we were not able to measure the Pareto fronts for Dota 5v5, and so we can only report the batch size used by the Dota team and the measured noise scale.

Results for the tasks described above were generally within a reasonable margin of state-of-the-art results, though we did not explicitly try to match SOTA or use special algorithmic or architectural tricks. Our goal was simply to confirm that we were in a reasonably well-performing regime that is typical of ML practice.

For the supervised learning and generative modeling tasks listed above, we have the option of using either training set or test set performance to compare different batch sizes. For the main results in this paper, we choose train set performance because it is what is directly predicted by our model, and because it is easier to measure in the presence of overfitting. The choice makes negligible difference in most of our experiments, either because they involve RL or because the datasets are large enough that we don't overfit. On the small datasets MNIST, CIFAR10, and SVHN, overfitting makes measurement of test error more difficult, but we do measure the test error behavior in Appendix E.3, and both the Pareto fronts and critical batch size generally do not change much.

The fact that the noise scale is consistent between well-tuned training runs suggests that the corresponding optimization trajectories are similar in some sense. In Appendix C we investigate this idea further and relate the noise scale to a characteristic 'temperature' of the training process.

**Model Size Dependence**

The definitions of the noise scales do not have any manifest dependence on the number of parameters in a model. We have conjectured that they will be roughly independent of model size at fixed values of the loss.

LSTM language models provide a natural test case, since LSTM sizes can be scaled up without qualitatively altering model architecture. As shown in Figure 8, the simple noise scale appears to be roughly independent of model size at a fixed value of the loss. However, due to their higher performance and lower achieved loss, larger models eventually reach larger noise scales than smaller models. We do not have specific hypotheses for how the noise scale should vary with model architecture, but interesting results along these lines were recently obtained [SLA$^+$18].

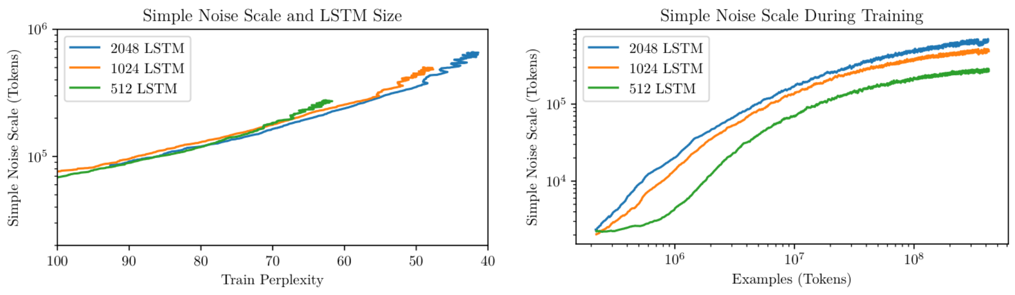


Figure 8: We show the relationship between training perplexity and the simple noise scale (left) for a range of LSTM sizes on the Billion Word dataset. These results show that *at fixed values of the loss, the noise scale does not depend significantly on model size*. On the right we show the simple noise scale during training, plotted in terms of the number of tokens processed. After processing a given number of examples, larger models will tend to have a larger noise scale, but only as a consequence of having achieved smaller loss.

## 4 Related Work

A great deal of prior work has studied large-batch training, investigated versions of the noise scale, explored adaptive batch size and learning rate schedules, and demonstrated that large batch training can be effective on specific datasets. We attempt to summarize this work below.

Recent papers have probed the limits of large batch training empirically, especially for ImageNet [GDG$^+$17, YZH$^+$17, JSH$^+$18], in some cases using layer-wise adaptive learning-rates [YGG17]. More recent work has demonstrated that large batch training can also be applied to RL [AAG$^+$18, BCD$^+$18, SA18, HQB$^+$18]. The use of second order optimization methods [BGM17] might increase the utility of data parallelism even further. A thorough review of large batch training and potential issues with generalization was provided in a very nice recent empirical study [SLA$^+$18] done in parallel with this work. [GVY$^+$18] also systematically studied large-batch training, though it did not tune the learning rate separately for each batch size.

Other recent work has explored the impact of gradient noise on optimization speed and batch size selection. [SZL13] connected gradient noise and the locally optimal step size to identify an adaptive learning rate. [MHB17] derived a sampling distribution for SGD, motivating our definition of 'temperature'. [SL17] connected this temperature to the critical batch size, though they predict a dependence on dataset size which we do not observe. [SZT17] identified a signal-dominated and noise-dominated phase of training. [SKYL17] showed that decaying the learning rate and increasing the batch size have the same effect, motivated by the SGD training temperature. ([DNG17] also suggested increasing learning rate and batch size together, but with different motivation.) [IES$^+$18] empirically investigated the role of gradient noise in reinforcement learning.

The gradient noise scale in particular has also been studied in earlier work to aid in batch size selection. The noise scale itself is used implicitly in basic statistical techniques for sample size selection (see e.g. [Wik, NIS]). [BCNW12] implicitly uses the gradient noise scale for a theoretical analysis of batch size selection. [BCN16, DYJG16, BRH16] propose adaptive sampling methods based on the gradient noise scale in the context of neural network optimization. [YPL$^+$17] analyzed the gradient noise scale for a particular class of functions and related it to the critical batch size, though it predicts a sharp change in learning speed with batch size rather than the smooth change we observe. [CWZ$^+$18] theoretically analyzed the dependence of the gradient noise scale on network width for shallow or linear networks, though they find inconsistent empirical results on neural networks. [MBB17] found a formula for the optimization speedup in terms of batch size resembling ours, though their critical batch size depends on smoothness parameters of the loss rather than directly gradient noise.

There has been a variety of work studying the Neural Network loss landscape and using it to draw conclusions about optimal training. Local properties of the loss landscape are not necessarily a good guide to overall optimal training [WRLG18]. The loss tends to be fairly smooth when interpolating between the start and end of training [GVS14]. But noise may be useful early in training [NVL$^+$15, YPL$^+$17], perhaps because it leads to minima that generalize better [KMN$^+$16].

A big-picture motivation for our work was to better understand the scaling of learning with computational and data resources; this question was addressed from the perspective of scaling the model size in [HNA$^+$17].

Our key contributions include connecting the gradient noise scale to the speed of optimization with a simple model, as well as systematically measuring the critical batch size and noise scale for a variety of tasks. We also clarify the role of the training temperature in SGD and propose an optimal batch size schedule.

## 5 Discussion

We have shown that the simplified gradient noise scale $\mathcal{B}_{\text{simple}}$ approximately predicts the actual point of diminishing return on batch size $\mathcal{B}_{\text{crit}}$ on diverse problems where these quantities vary by six orders of magnitude. Furthermore, the tradeoff curve between total compute and optimization steps associated with changing the batch size has roughly the hyperbolic form predicted by our theory. Finally, our theory also roughly predicts how the optimal learning rate scales with batch size, although its predictions are not as precise.

What does the validity of this theory mean, and in what way is it useful? At the level of a given task, it allows us to use the noise scale from a single run (even an only partially complete run with much smaller batch size, though see caveats about learning rate tuning in the appendix) to estimate the largest useful batch size, and thus reduces the extensive hyperparameter searches that are necessary to find this batch size by trial and error. It also tells us to expect that larger batch sizes will show diminishing returns in a predictable way that has the same form regardless of the task.

Across tasks, it tells us that the largest useful batch size for a task is likely to be correlated to informal notions of the "complexity" of the task, because the noise scale essentially measures how diverse the data is (as seen by the model), which is one aspect of task complexity.

We have argued that a specific formula characterizes the time/compute tradeoff between optimization steps and total data processed in neural network training:

$$\left(\frac{\text{Optimization Steps}}{\text{Min Steps}} - 1\right)\left(\frac{\text{Data Examples}}{\text{Min Examples}} - 1\right) = 1 \tag{5.1}$$

From this relation we can identify a critical value of the batch size when training to a given value of the loss

$$\mathcal{B}_{\text{crit}}(\text{Loss}) = \frac{\text{Min Examples}}{\text{Min Steps}}$$

Training at this critical batch size provides a natural compromise between time and compute, as we take only twice the minimum number of optimization steps and use only twice the minimum amount of data. The critical batch size represents a turning point, so that for $B > \mathcal{B}_{\text{crit}}$ there are diminishing returns from greater data parallelism.

Our main goal was to provide a simple way to predict $\mathcal{B}_{\text{crit}}$. We have shown that it can be estimated as

$$\mathcal{B}_{\text{crit}} \approx \mathcal{B}_{\text{simple}} \tag{5.2}$$

where the easily-measured $\mathcal{B}_{\text{simple}}$ is the ratio of the gradient variance to its squared mean. Theoretical arguments suggest that a more refined quantity, the Hessian-weighted $\mathcal{B}_{\text{noise}}$ of Equation 2.8, may provide an even better[10] estimate of $\mathcal{B}_{\text{crit}}$.

The tradeoff curve of Equation 5.1 provides a remarkably good fit across datasets, models, and optimizers, and the approximate equality of $\mathcal{B}_{\text{crit}}$ and $\mathcal{B}_{\text{simple}}$ holds even as both quantities vary greatly between tasks and training regimes. We have established that as anticipated, both $\mathcal{B}_{\text{crit}}$ and $\mathcal{B}_{\text{simple}}$ tend to increase significantly during training, that they are larger for more complex tasks, and that they are roughly independent of model

[10]We have also investigated using gradients preconditioned by the Adam optimizer; the results were mixed.

size (for LSTMs) at fixed values of the loss. We also saw that image classification has a significantly larger per-image noise scale as compared to generative models training on the same dataset, a fact that could have interesting implications for model-based RL. In the case of RL, while the noise scale for Dota was roughly a thousand times larger than that of Atari, the total number of optimization steps needed to train a Dota agent is not so much larger [BCD$^+$18]. Perhaps this suggests that much of the additional compute needed to train more powerful models will be parallelizable.

While $\mathcal{B}_{\text{simple}}$ roughly matches $\mathcal{B}_{\text{crit}}$ for all datasets, the ratio $\mathcal{B}_{\text{simple}}/\mathcal{B}_{\text{crit}}$ can vary by about an order of magnitude between tasks. This may not be so surprising, since $\mathcal{B}_{\text{simple}}$ does not take into account Hessian conditioning or global aspects of the loss landscape. But it would be very interesting to obtain a better understanding of this ratio. It was smaller than one for the Autoencoder, VAE, and for Dota 1v1, roughly equal to one for LSTMs, and greater than one for both image classification tasks and Atari, and we lack an explanation for these variations. It would certainly be interesting to study this ratio in other classes of models, and to further explore the behavior of generative models.

Due to its crucial role in data-parallelism, we have focused on the batch size $B$, presuming that the learning rate or effective 'temperature' will be optimized after $B$ has been chosen. And our theoretical treatment focused on a specific point in the loss landscape, ignoring issues such as the relationship between early and late training and the necessity of a 'warm-up' period. It would be interesting to address these issues, particularly insofar as they may provide motivation for adaptive batch sizes.

**Acknowledgements**

We are grateful to Paul Christiano for initial ideas and discussions about this project. We would like to thank the other members of OpenAI for discussions and help with this project, including Josh Achiam, Danny Hernandez, Geoffrey Irving, Alec Radford, Alex Ray, John Schulman, Jeff Wu, and Daniel Ziegler. We would also like to thank Chris Berner, Chris Hesse, and Eric Sigler for their work on our training infrastructure. We thank Joel Hestness, Heewoo Jun, Jaehoon Lee, and Aleksander Madry for feedback on drafts of this paper. JK would also like to thank Ethan Dyer for discussions.

## A Methods

### A.1 Unbiased Estimate of the Simple Noise Scale with No Overhead

In this section, we describe a method for measuring the noise scale that comes essentially for free in a data-parallel training environment.

We estimate the noise scale by comparing the norm of the gradient for different batch sizes. From Equation 2.2, the expected gradient norm for a batch of size $B$ is given by:

$$\mathbb{E}\left[|G_{\text{est}}|^2\right] = |G|^2 + \frac{1}{B}\text{tr}(\Sigma). \tag{A.1}$$

Given estimates of $|G_{\text{est}}|^2$ for both $B = B_{\text{small}}$ and $B = B_{\text{big}}$, we can obtain unbiased estimates $|\mathcal{G}|^2$ and $\mathcal{S}$ for $|G|^2$ and $\text{tr}(\Sigma)$, respectively:

$$\begin{aligned} |\mathcal{G}|^2 &\equiv \frac{1}{B_{\text{big}} - B_{\text{small}}}\left(B_{\text{big}}|G_{B_{\text{big}}}|^2 - B_{\text{small}}|G_{B_{\text{small}}}|^2\right) \\ \mathcal{S} &\equiv \frac{1}{1/B_{\text{small}} - 1/B_{\text{big}}}\left(|G_{B_{\text{small}}}|^2 - |G_{B_{\text{big}}}|^2\right). \end{aligned} \tag{A.2}$$

We can verify with Equation A.1 that $\mathbb{E}\left[|\mathcal{G}|^2\right] = |G|^2$ and $\mathbb{E}\left[\mathcal{S}\right] = \text{tr}(\Sigma)$.[11]

Note that the ratio $\mathcal{S}/|\mathcal{G}|^2$ is not an unbiased estimator for $\mathcal{B}_{\text{noise}}$[12]. It is possible to correct for this bias, but to minimize complexity we instead ensure that $|\mathcal{G}|^2$ has relatively low variance by averaging over many batches. This is especially important due to the precise cancellation involved in the definition of $|\mathcal{G}|^2$.

When training a model using a data parallel method, we can compute $|G_{B_{\text{small}}}|^2$ and $|G_{B_{\text{big}}}|^2$ with minimal effort by computing the norm of gradient before and after averaging between devices. In that case $B_{\text{small}}$ is the "local" batch size before averaging, and $B_{\text{big}}$ is the "global" batch size after averaging.

In practice, to account for the noisiness of $|\mathcal{G}|^2$ when computed this way, we calculate $|\mathcal{G}|^2$ and $\mathcal{S}$ on every training step and use their values to compute separate exponentially-weighted moving averages. We tune the exponential decay parameters so that the estimates are stable. Then, the ratio of the moving averages provides a good estimate of the noise scale.

In our experiments we measure and report the noise scale during training for a single run with a well-optimized learning rate. Note that while the noise scale measurement is consistent between runs at different batch sizes, it is not consistent at different learning rates (see Appendix C). So, it is important to use a run with a well-tuned learning rate in order to get a meaningful noise scale measurement.

### A.2 Systematic Searches Over Batch Sizes

When doing systematic measurements of how performance scales with batch size (Pareto fronts), we separately tune the learning rate at each batch size, in order to approximate the ideal batch scaling curve as closely as possible. We tune the learning rate via the following procedure. For each task, we performed a coarse grid search over both batch size and learning rate to determine reasonable bounds for a fine-grained search. The central value typically followed the form

$$\epsilon_{\text{central}}(B) = \frac{\epsilon_*}{(1 + B_*/B)^{\alpha}}, \tag{A.3}$$

where $\alpha = 1$ for SGD or momentum, and $0.5 < \alpha < 1$ for Adam [KB14] or RMSProp. Then, we performed an independent grid search for each batch size centered at $\epsilon_{\text{central}}$, expanding the bounds of the search if the best value was on the edge of the range.

[11] Note that when $B_{\text{small}} = 1$ and $B_{\text{big}} = n$, this becomes the familiar Bessel correction $\frac{n}{n-1}$ to the sample variance.

[12] In fact $\mathbb{E}\left[x/y\right] \geq \mathbb{E}\left[x\right]/\mathbb{E}\left[y\right]$ in general for positive variables, see e.g. `https://en.wikipedia.org/wiki/Ratio_estimator` for details.

We explain the motivation for Equation A.3 in Appendix E.2. But regardless of the theoretical motivations, we have found that this scaling rule provides a reasonable starting point for grid searches, though we are not suggesting that they produce precisely optimized learning rates.

### A.3 Pareto Front Measurements

To produce the Pareto front plots, and thus to measure the important parameter $\mathcal{B}_{\text{crit}}$ for a given dataset and optimizer, we begin by performing a grid search over batch sizes and learning rates, as described in Appendix A.2. With that data in hand, we fix a list of goal values – either loss, perplexity, or game-score. For example for SVHN in Figure 7 we chose the training classification error values $[0.2, 0.1, 0.07, 0.05, 0.035, 0.025, 0.015, 0.004]$ as the goals. These were generally chosen to provide a variety of evenly spaced Pareto fronts indicative of optimization progress.

Then for each value of the goal, and for each value of the batch size, we identified the number of optimization steps and examples processed for the run (among those in the grid search) that achieved that goal most quickly. These optimal runs are the data points on the Pareto front plots. Note that at fixed batch size, different values of the learning rate might be optimal for different values of the goal (this was certainly the case for LSTMs on Billion Word, for example). Next, for each value of the goal, we used the optimal runs at each value of the batch size to fit Equation 2.11 to the relation between examples processed and optimization steps. Note that we performed the fits and extracted the errors in log-space. This was how we produced the lines on the Pareto front plots.

Finally, given this fit, we directly measured $\mathcal{B}_{\text{crit}} = \frac{E_{\min}}{S_{\min}}$ for each value of the goal, as well as the standard error in this quantity. This was how we produced the 'Noise Scale Comparison' plots, where we compared $\mathcal{B}_{\text{crit}}$ to $\mathcal{B}_{\text{simple}}$. Errors in $\mathcal{B}_{\text{crit}}$ are standard errors from the fit to Equation 2.11. When we report an overall number for $\mathcal{B}_{\text{crit}}$ for a given dataset and optimizer, we are averaging over optimization steps throughout training.

Note that it can be difficult to determine at what point in a training run the model's performance reaches the specified target. For example, the loss may oscillate significantly, entering and exiting the target region multiple times. To remedy this issue, we smooth the loss using an exponentially-weighted moving average before checking whether it has reached the target. The decay parameter of this moving average can affect results noticeably. Though we choose this parameter by hand based on the noisiness of the model's performance, this could be automated using an adaptive smoothing algorithm.

### A.4 Details of Learning Tasks

We train a variety of architectures on a variety of ML tasks described below. We use either basic stochastic gradient descent (SGD), SGD with momentum [SMDH13], or the Adam optimizer [KB14] unless otherwise specified. We measure and report the noise scale $\mathcal{B}_{\text{simple}}$ during training for a single run of each task with a well-optimized learning rate.

#### A.4.1 Classification

For image classification, we use the following datasets:

- MNIST handwritten digits [LC10]
- Street View House Numbers (SVHN) [NWC$^+$11]
- CIFAR10 [Kri09]
- ImageNet [DDS$^+$09]

For CIFAR10 and ImageNet classification, we use Residual Networks [HZRS15] of size 32 and 50 respectively, based on the TensorFlow Models implementation [Goo]. All hyperparameters are unchanged aside from the learning rate schedule; Instead of decaying the learning rate by a factor of 10 at specified epochs, we decay by a factor of 10 when the training classification error (appropriately smoothed) reaches 0.487, 0.312, and 0.229. For MNIST and SVHN, we use a simple deep network with two sets of convolutional and pooling layers (32 and 64 filters, respectively, with 5x5 filters), one fully-connected hidden layer with 1024 units, and a final dropout layer with dropout rate of 0.4.

We train MNIST models using SGD, SVHN with both SGD and Adam [KB14] (with the default parameter settings momentum $= 0.9$, $\beta_2 = 0.999$), and CIFAR10 and ImageNet with momentum [SMDH13] (with momentum $= 0.9$).

### A.4.2 Reinforcement Learning

For reinforcement learning, we use the following tasks via OpenAI Gym [BCP$^+$16]:

- Atari Arcade Learning Environment [BNVB12]
- Dota 1v1 and 5v5 [BCD$^+$18]

For Atari, we use A2C [MBM$^+$16] with a pure convolutional policy, adapted from OpenAI Baselines [DHK$^+$17]. We train using RMSProp with $\alpha = 0.99$ and $\epsilon = 10^{-5}$. We roll out the environments 5 steps at a time, and vary the batch size by varying the number of environments running parallel. At the beginning of training, we randomly step each parallel environment by a random number of steps up to 500, as suggested in [SA18].

As described in [BCD$^+$18] for Dota an asynchronous version of PPO [SWD$^+$17] was used. The TrueSkill metric [HMG07] was used to measure the skill of an agent. Given the fact that the OpenAI Five effort is ongoing, the values for TrueSkill reported in this paper are incomparable with those in [BCD$^+$18]; on this paper's scale, TrueSkill 50 is roughly the level of the best semi-pro players.

### A.4.3 Generative and Language Modeling

For language modeling, we train a size-2048 LSTM [HS97] on the One Billion Word Benchmark corpus [CMS$^+$13], using byte pair encoding (BPE) [SHB15] with a vocabulary of size 40,000 and a 512-dimensional embedding space. The LSTMs were trained with Adam using momentum $0.5$, without dropout, with the gradients clipped to norm 10, and with 20-token sequences. For both training and evaluation LSTM cell states were reset to zero between samples, and so we have reported perplexity for the last token of the 20-token sequences. We chose to report the batch size in tokens (rather than sequences) because we have found that when the number of sequences and the sequence lengths are varied, both $\mathcal{B}_{\text{simple}}$ and $\mathcal{B}_{\text{crit}}$ depend predominantly on the total number of tokens.

We also trained 1024 and 512 size LSTMs for model size comparison; for the last we used a smaller $256$-dimensional embedding space. The model size comparison training runs were conducted with a batch size of $1024$ and Adam learning rate of $0.0007$. The learning rates were chosen from a grid search, which showed that the optimal learning rate did not have a significant dependence on model size.

For generative image modeling, we train a Variational Autoencoder [KW13] using the InfoGAN architecture [CDH$^+$16] (see their appendix C.2) on the SVHN dataset. Since VAEs introduce additional stochasticity beyond gradient noise, we also provide training data on a simple autoencoder with the same architecture.

## B Results for All Tasks

In Figures 9, 10, 11, 12, 13, and 14, we display the results of a series of training runs for for classification, reinforcement learning, and generative modeling tasks. On the left, we show tradeoff curves between compute-efficiency and time-efficiency. Each point on each tradeoff curve represents the number of optimizer steps and processed training examples necessary to reach a given level of performance for a particular training run. Fits to the prediction of Equation 2.11 are shown. On the right, we compare the critical batch size, defined as the point where training is within 50% of maximum efficiency in terms of both compute power and speed, and compare to the simple noise scale $\mathcal{B}_{\text{simple}}$ of Equation 2.9 and the true noise scale $\mathcal{B}_{\text{noise}}$ of 2.8, when available. The results are summarized in Figure 4 and table 1.

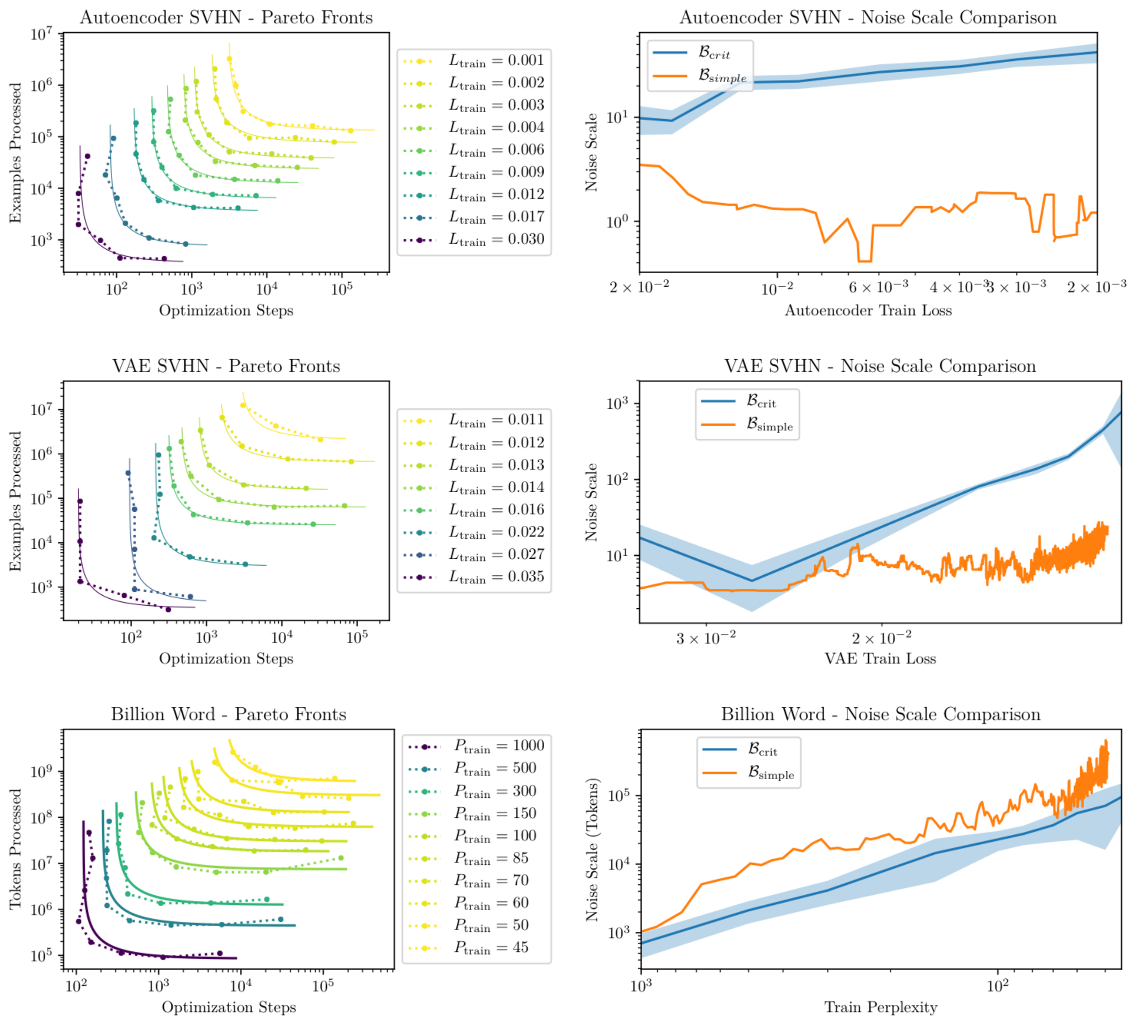


Figure 9: Scaling behavior of generative and language modeling tasks.

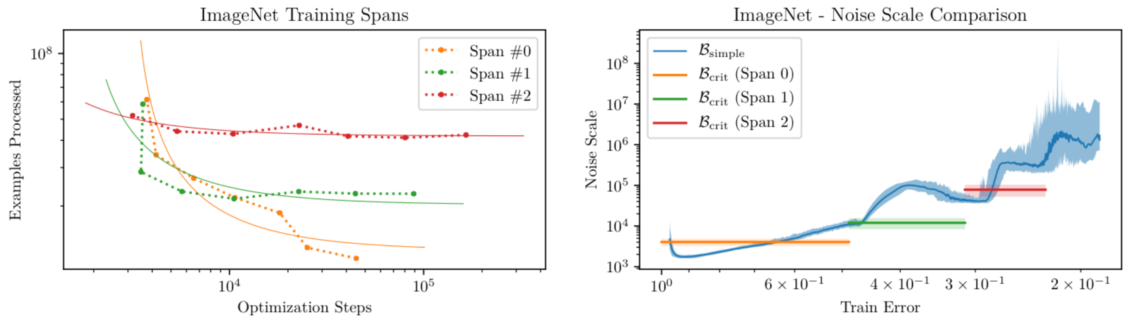


Figure 10: For ImageNet, the typical training schedule decays the learning rate by a factor of 10 at 30, 60, and 80 epochs [HZRS15, GDG+17]. To provide a fair comparison between batch sizes, we instead decay by a factor of 10 when the training classification error reaches 0.487, 0.312, and 0.229. We display Pareto fronts and compute the critical batch size separately for each span.

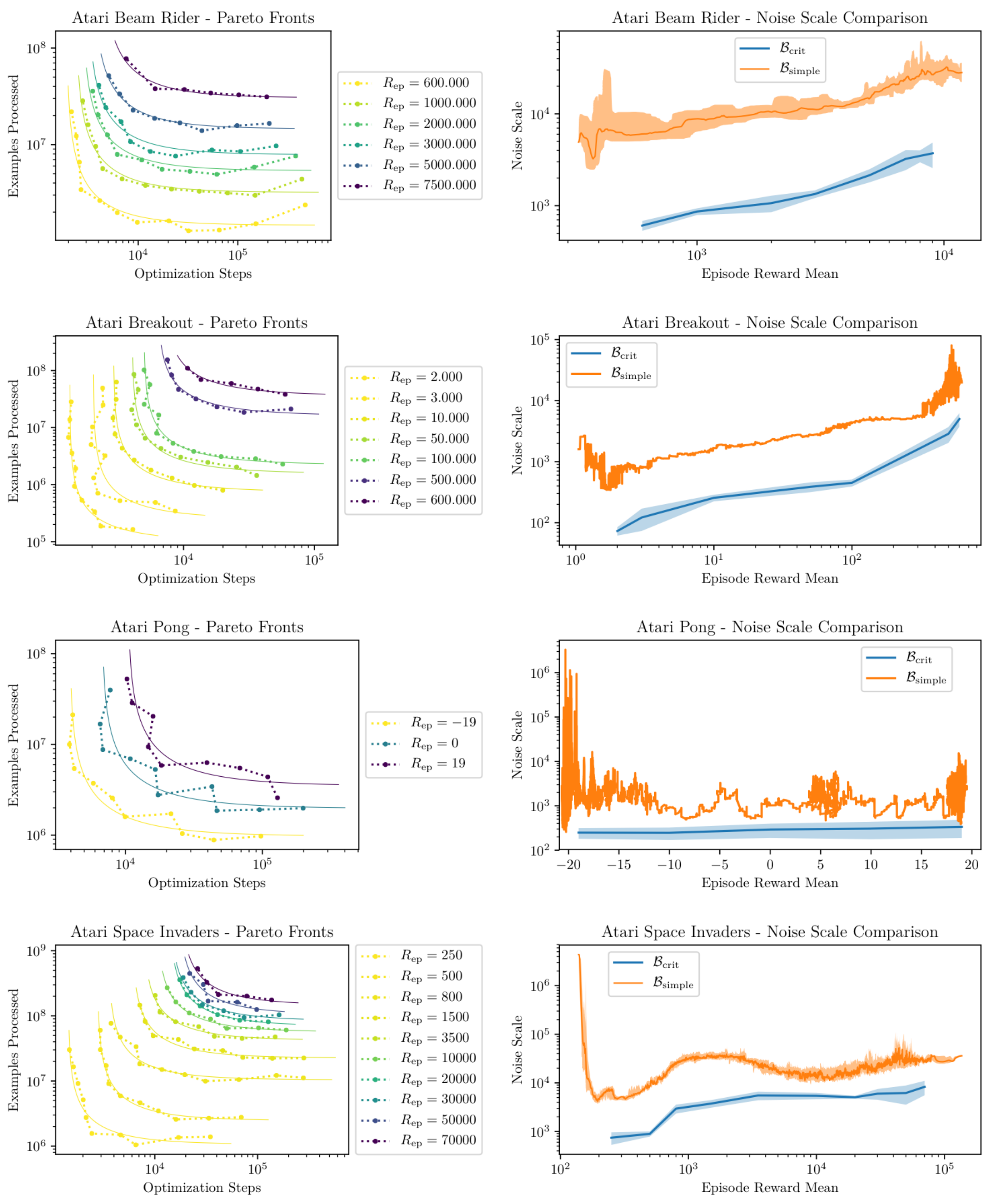


Figure 11: Scaling behavior of Atari tasks (Beam Rider, Breakout, Pong, and Space Invaders) trained with A2C.

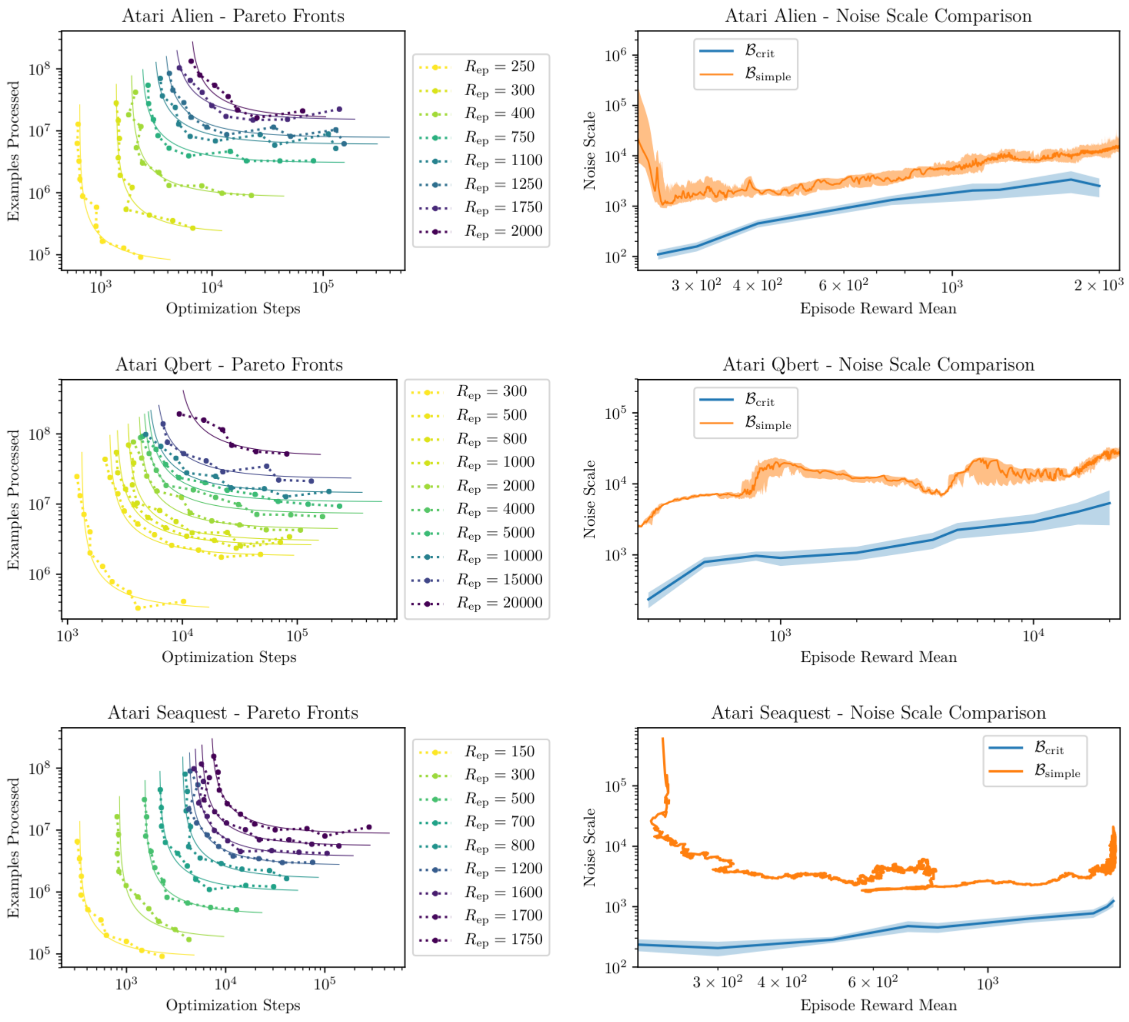


Figure 12: Scaling behavior of more Atari tasks (Alien, Qbert, and Seaquest) trained with A2C.

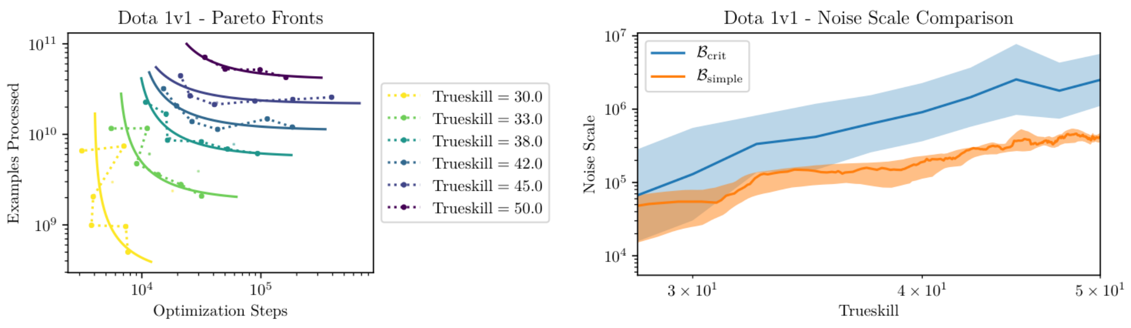


Figure 13: Scaling behavior for Dota 1v1 [Ope17] trained to top-level pro performance.

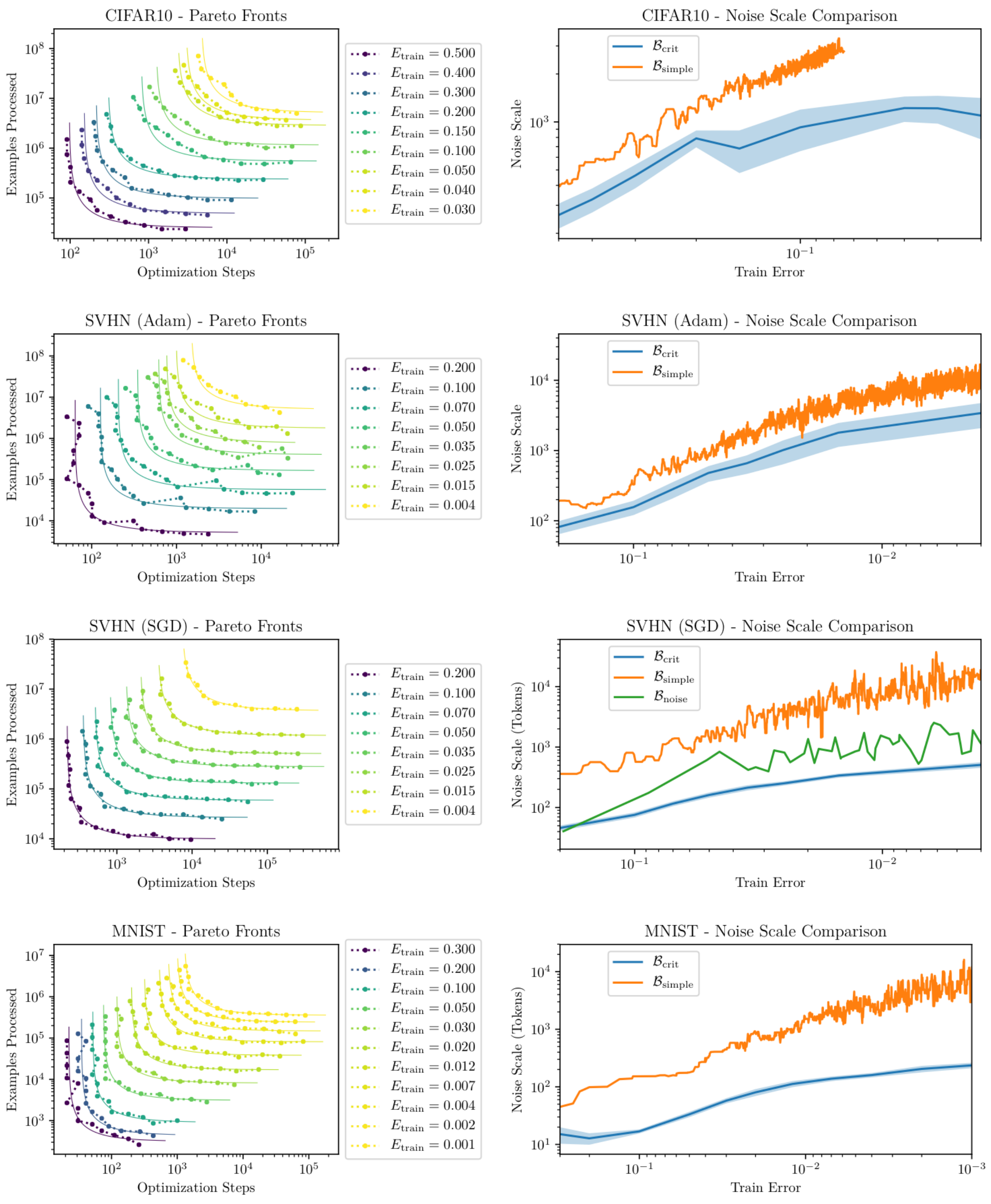


Figure 14: Scaling behavior of image classification tasks.

| | Critical Batch Size | | Simple Noise Scale | |
|---|---|---|---|---|
| | Start | Average | Start | Average |
| **Image Classification:** | | | | |
| MNIST | 20 | 200 | 50 | 900 |
| SVHN | 50 | 500 | 300 | 4,000 |
| CIFAR10 | 300 | 900 | 400 | 2,000 |
| ImageNet | 1,000 | 15,000 | 4,000 | 30,000 |
| **Generative and Language Modeling:** | | | | |
| Autoencoder (SVHN) | 10 | 40 | 2 | 2 |
| Variational Autoencoder (SVHN) | 10 | 200 | 10 | 10 |
| Billion Word (per token) | 700 | 100,000 | 1000 | 150,000 |
| **Reinforcement Learning:** | | | | |
| Atari (per frame) | 100 - 1,000 | 400 - 8,000 | 100-1,000 | 1,000-20,000 |
| Dota 1v1 (per frame) | 50,000 | 3,000,000 | 100,000 | 300,000 |
| Dota 5v5 (per frame) | (not measured) | >8,000,000 (est.) | 100,000 | 24,000,000 |

Table 1: We report the simple noise scale, both early in training and averaged over a training run, as well as the critical batch size, both early in the run and at the end of the run. The noise scale provides a good estimate for the critical batch size throughout training. Batch sizes reported in number of images, tokens (for language models), or observations (for games). These data are summarized in Figure 4.

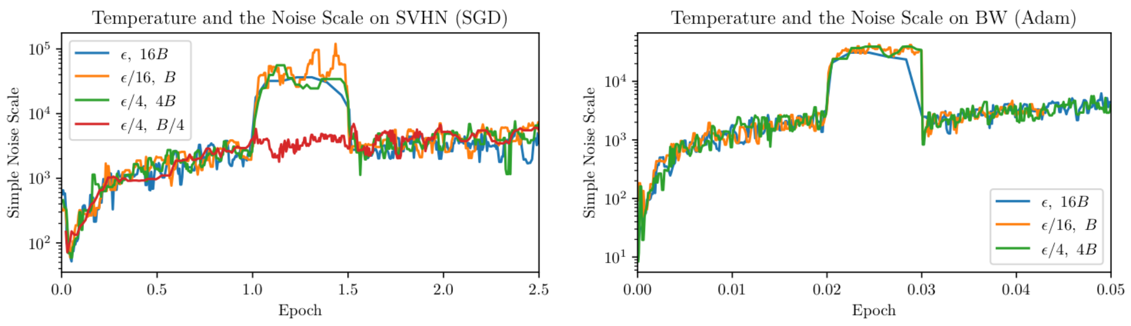


Figure 15: The noise scale is proportional to the inverse temperature. On the left we display results for SVHN optimized via SGD, while on the right we have an LSTM on the Billion Word dataset optimized via Adam. For each of the three curves, we modified either the learning rate $\epsilon$, the batch size $B$, or both, so that the temperature $\frac{\epsilon}{B}$ was decreased by a factor of 16 between epochs 1 and 1.5 (SVHN) or 0.02 and 0.03 (BW). In all cases we see that the simple noise scale increased by a factor of 16, then returned to roughly its original value once $\epsilon$, $B$ were reset.

## C Temperature and the Noise Scale

The noise scale measured during neural network training could depend on a variety of hyperparameters, such as the learning rate $\epsilon$ or momentum. However, we have empirically found that noise scale primarily depends on $\epsilon$ and $B$ roughly through the ratio

$$T(\epsilon, B) \equiv \frac{\epsilon}{\epsilon_{\text{max}}(B)}, \tag{C.1}$$

which we refer to as the 'temperature' of training. The terminology reflects an idea of the loss as a potential energy function, so that high temperature training explores a larger range of energies.

In the case of pure SGD it is approximated by $T \approx \epsilon/B$ in the small batch regime. Our definition of $T$ can then be obtained from a toy model of a quadratic loss, which is described below. In that case one can show explicitly [MHB17] that the equilibrium distribution of gradients is characterized by this temperature[13].

In equilibrium, the noise scales vary in proportion to the inverse temperature, so that

$$\mathcal{B}_{\text{noise}} \propto \mathcal{B}_{\text{simple}} \propto \frac{1}{T}. \tag{C.2}$$

It may seem surprising that higher temperature results in a smaller noise scale. The intuition is that at larger $T$ the neural network parameters are further from the minimum of the loss, or higher up the 'walls' of the potential, so that the gradient magnitude is larger relative to the variance.

Of course the loss landscape will be much more complicated than this toy model, but we have also observed that this scaling rule provides a good empirical rule of thumb, even away from pure SGD. In particular, when we decay the learning rate $\epsilon$ by a constant factor, we often find that the noise scale grows by roughly the same factor. ImageNet training provides an example in Figure 10. A more direct investigation of the relation between $\mathcal{B}_{\text{simple}}$ and $T$ is provided in Figure 15.

Since the noise depends primarily on the training temperature, and well-tuned training runs should have the same temperature at different batch sizes, the measured noise scale will also be consistent between optimally-tuned runs at different batch sizes.[14]. The noise scale then depends only on the temperature and the loss.

[13]This definition can also be motivated by the empirical results of [SKYL17], which show that decaying the learning rate and increasing the batch size by the same factor have the same effect on training in the small-batch regime.

[14]This suggests that we can use the noise scale to *define* the temperature via Equation C.2. Then, once we have tuned the learning rate and measured the noise scale at small batch size, we can tune the learning rate at larger batch sizes to the noise scale constant. Though we have not investigated this idea thoroughly, it could significantly simplify the problem of learning rate tuning at large batch size.

To summarize, the noise scale does *not* provide an optimal training temperature schedule, but it instead prescribes an optimal batch size at any *given* temperature.

### A Toy Model for the Temperature

Now let us consider a simple explanation for the behavior of the noise scale in response to changes in the learning rate $\epsilon$ and batch size $B$. We start by approximating the loss as locally quadratic:

$$L(\theta) = \frac{1}{2}\theta^T H\theta + \text{const.}$$

where we set $\theta = 0$ at the minimum without loss of generality. To compute the noise scale, we need a model for the gradient covariance matrix $\Sigma$. A simple model appearing in [SZL13] suggests treating the per-example loss $L_i$ as a shifted version of the true loss, $L_i(\theta) = L(\theta - c_i)$, where $c_i$ is a random variable with mean zero and covariance matrix $\Sigma_c$. The gradient covariance matrix is then given by $\Sigma = H\Sigma_c H$, which is independent of $\theta$. The average gradient itself is given by $G = H\theta$, with $\theta$ changing in response to $\epsilon$ or $B$. As shown in [MHB17] over sufficiently long times SGD[15] will approximately sample $\theta$ from the distribution

$$p_{\text{SGD}}(\theta) \propto \exp\left[-\frac{1}{2}\theta^T M^{-1}\theta\right]$$

where the matrix $M$ satisfies

$$MH + HM = \frac{\epsilon}{B}\Sigma.$$

From these results, we can estimate the noise scale:

$$\mathcal{B}_{\text{simple}} = \frac{\text{tr}(\Sigma)}{|G|^2} \approx \frac{B}{\epsilon}\frac{\text{tr}(\Sigma)}{\text{tr}(H^2\Sigma)}$$
$$\mathcal{B}_{\text{noise}} = \frac{\text{tr}(\Sigma)H}{G^T HG} \approx \frac{B}{\epsilon}\frac{\text{tr}(H\Sigma)}{\text{tr}(H^3\Sigma)}$$

So, in this model, the noise scale is expected to increase as we decrease the learning rate or increase the batch size. We also expect that scaling the learning rate and batch size together should leave the noise scale unchanged.

When $B \ll \mathcal{B}_{\text{simple}}$, the ratio $\frac{\epsilon}{B}$ plays the role of a "temperature". Since our analysis was only based on a toy model optimized using pure SGD, one might not expect it to work very well in practice. However, as shown in Figure 15, we have found that it provides a quite accurate model of the dependence of the noise scale on $\epsilon$ and $B$ during neural network training, even when using the Adam[16] optimizer. For these tests, on SVHN we used an initial $(\epsilon, B) = (0.18, 128)$ while for billion word results we used $(\epsilon, B) = (6 \times 10^{-4}, 128)$.

Note that this result relies on the assumption that the optimizer has approached an effective equilibrium. We expect the equilibration timescale to be larger in the directions of low curvature, so that this effect will be strongest when the gradient points mostly in the large-curvature directions of the Hessian. It would be interesting to investigate the timescale for equilibration.

## D Dynamically Varying the Batch Size

As one can see from Figure 4 and Section 3, both the measured $\mathcal{B}_{\text{noise}}$ and $\mathcal{B}_{\text{simple}}$, as well as the empirical $\mathcal{B}_{\text{crit}}$ fit to Equation 2.11 all increase by at least an order of magnitude during training. Thus its natural to ask if we should expect to improve efficiency by dynamically scaling the batch size $B$ in response. We will see that the predicted gains are relatively modest unless the $\mathcal{B}_{crit}$ changes greatly during training, although preliminary empirical tests suggest the benefits may be larger than predicted.

[15]With momentum, the same statements hold with $\epsilon$ replaced by $\epsilon/(1-m)$.

[16]Note that with $\beta_2 = 0.999$ the Adam variance accumulators would take of order $\sim 1000$ steps to fully react. On the right in Figure 15 we changed $\epsilon$ and $B$ for 0.01 epochs, corresponding to between 100 and 1500 optimizer steps.

## D.1 Theory

Consider a single full-batch optimizer step, over which the loss increases by an amount $\delta L$. If we instead use a batch of size $B$, it will take $\delta S = 1 + \frac{\mathcal{B}}{B}$ optimizer steps and $\delta E = B\delta S$ training examples to make the same amount of progress, where $\mathcal{B}$ is the noise scale. Over a full training run, the total number of steps and data examples processed can be written as

$$\begin{aligned} S &= \int \left(1 + \frac{\mathcal{B}(s)}{B(s)}\right) ds \\ E &= \int \left(\mathcal{B}(s) + B(s)\right) ds \end{aligned} \tag{D.1}$$

where we parameterize the training trajectory by the number $s$ of full-batch optimizer steps (we abbreviated $S_{\min}$ above to $s$ for notational simplicity).

The question is how to optimally distribute the training examples over the full training trajectory. At each point along the trajectory, we have the choice of trading examples for optimizer steps by increasing or decreasing the batch size. This "exchange rate" between examples and steps is

$$r = -\frac{\frac{d}{dB}\delta E}{\frac{d}{dB}\delta S} = \frac{B^2(s)}{\mathcal{B}(s)}. \tag{D.2}$$

If the distribution of training examples (and hence the batch size schedule) is optimal, then transferring examples from one part of training to another should not save any optimization steps. This means that *the exchange rate $r$ should be constant throughout training*. Thus the batch size should be varied in proportion with the square root of the noise scale:

$$B(s) = \sqrt{r\mathcal{B}(s)}. \tag{D.3}$$

We can determine the resultant Pareto front parameterizing the tradeoff between training cost and time by inserting Equation D.3 into Equation D.1 and eliminating the exchange rate[17]

$$\frac{S_{\text{tot}}}{S_{\min}} - 1 = \gamma \left(\frac{E_{\text{tot}}}{E_{\min}} - 1\right)^{-1}, \tag{D.4}$$

where we define $S_{\min} \equiv \int ds$ and $E_{\min} \equiv \int \mathcal{B} ds$ to be the minimum possible number of optimizer steps and training examples needed to reach the desired level of performance, obtained by inserting $B \gg \mathcal{B}$ and $B \ll \mathcal{B}$ respectively into D.3. We also define

$$\gamma \quad \equiv \quad \frac{\left(\int \sqrt{\mathcal{B}} ds\right)^2}{S_{\min} E_{\min}}, \tag{D.5}$$

which parameterizes the amount of variation of the noise scale over the course of training. When the noise scale is constant $\gamma = 1$ and there is no benefit from using an adaptive batch size; more variation in $\mathcal{B}$ pushes $\gamma$ closer to 0, yielding a corresponding predicted improvement in the Pareto front.[18] Note that since $\gamma$ involves the variation in the square root of $\mathcal{B}$, practically speaking $\mathcal{B}$ must vary quite a bit during training for adaptive batch sizes to provide efficiency benefits via these effects. Adaptive batch sizes may also have other benefits, such as replacing adaptive learning rates [SKYL17] and managing the proportion of gradient noise during training.

[17]The exchange rate $r$ is a free parameter. It can be chosen according to preference from the value of training time vs compute. There is also the fairly natural choice $r = \frac{E_{\min}}{S_{\min}}$ at which we have $\frac{S_{\text{tot}}}{S_{\min}} = \frac{E_{\text{tot}}}{E_{\min}} = 1 + \sqrt{\gamma}$, so that cost-efficiency and time-efficiency are both within the same factor of optimal, corresponding to the turning point in Figure 16.

[18]To see explicitly the dependence of $\gamma$ on the variability of the noise scale, we can rewrite it as $\gamma = \frac{\mathbb{E}\left[\sqrt{\mathcal{B}}\right]^2}{\mathbb{E}[\mathcal{B}]} = \frac{1}{1+\sigma^2_{\sqrt{\mathcal{B}}}/\mathbb{E}\left[\sqrt{\mathcal{B}}\right]^2}$, where the expectation is over a training run, weighting each full-batch step equally.

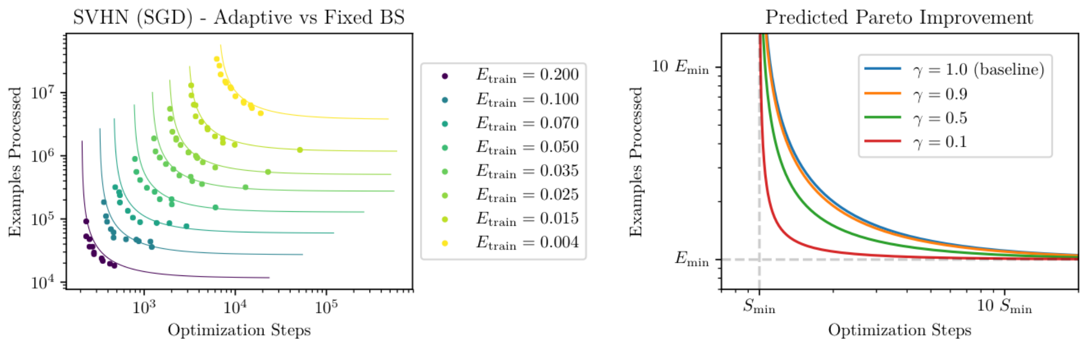


Figure 16: **Left:** We compare training using an adaptive batch size (data points) to the hyperbolic fit to Pareto fronts at fixed batch size (lines). We note a modest but visible improvement to training efficiency. Adaptive batch sizes appear to decrease the minimum number of optimization steps $S_{\rm min}$, which was not anticipated by theoretical analysis. **Right:** Depending on the degree to which the noise scale varies over training, we can predict the potential Pareto improvement from using an adaptive batch size.

### D.2 An SVHN Case Study

We have argued that a batch size of order the noise scale can simultaneously optimize data parallelism and total resource use. We have also shown that the noise scale tends to grow quite significantly during training. This suggests that one can further optimize resource use by adaptively scaling[19] the batch size with the noise scale as training progresses, as discussed above.

For adaptive batch training, we can follow a simple and pragmatic procedure and dynamically set

$$B = \sqrt{r\mathcal{B}_{\rm simple}}, \tag{D.6}$$

with $\mathcal{B}_{\rm simple}$ measured periodically during training. The results from dynamic batch training with this procedure and various values of $r$ are compared to fixed batch size training in Figure 16. We see that adaptive training produces a modest[20] efficiency benefit.

We can combine our fixed batch size results with theoretical analysis to predict the magnitude of efficiency gains that we should expect from adaptive batch size training. We displayed $\mathcal{B}_{\rm crit}$ for fixed batch training of SVHN in Figure 7. We have found that these results are fit very well by $\mathcal{B}_{\rm crit}(s) \approx 10\sqrt{s}$, where $s$ is the number of steps taken in the limit of very large batch training. Using Equation D.5, we would predict the quite modest efficiency gain of

$$\gamma = \frac{\left(\int ds \sqrt[4]{s}\right)^2}{s \int ds \sqrt{s}} = \frac{24}{25} \tag{D.7}$$

or around $4\%$. The benefits visible in Figure 16 in some cases appear too large to be fully explained by this analysis.

In particular, the adaptive batch size seems to benefit training in the regime of large batch size, decreasing the minimum number of optimization steps $S_{\rm min}$. However, our theoretical analysis would predict negligible benefits at large $E_{\rm tot}/S_{\rm tot}$. This may be due to the fact that the adaptive BS schedule also 'warms up' the learning rate, or it may be an effect of a larger and more consistent proportion of gradient noise during training. It would be interesting to disentangle these and other factors in future work, and to study adaptive batch size training on other datasets.

[19]This may have an additional advantage compared to training with a fixed, large batch size: it allows for a constant proportion of gradient noise during training, and some have argued [KMN+16, HHS17] that noise benefits generalization.

[20]It's challenging to provide a fair comparison between fixed and adaptive batch size training. Here we determined a roughly optimal relation $\epsilon = \frac{0.27B}{96+B}$ between learning rate $\epsilon$ and $B$ for fixed batch size training, and used this same function to determine the learning rate for both fixed and adaptive batch size training runs. This meant the adaptive batch size training used a corresponding adaptive learning rate. We did not experiment with learning rate schedules.

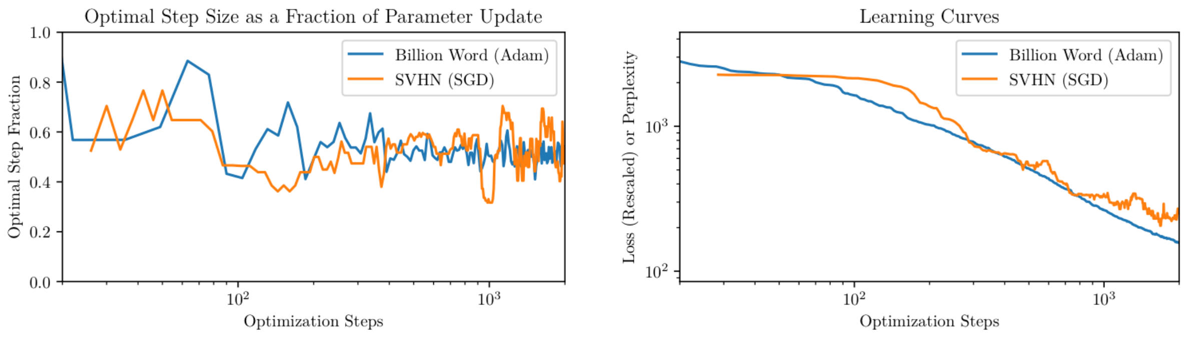


Figure 17: Left: This figure shows the magnitude of the optimal step size in the direction of the parameter update divided by the magnitude of the actual update. Optimal step sizes are determined by a line search of the loss. We show training of two quite different models with different optimizers – an LSTM trained with Adam (momentum = $0.5$) on Billion Word, and a CNN trained on SVHN with SGD. In both cases, training converges to an approximate steady state where the average update is about twice the optimal update. Right: Learning curves included to clarify that this phenomenon is not due to the cessation of learning.

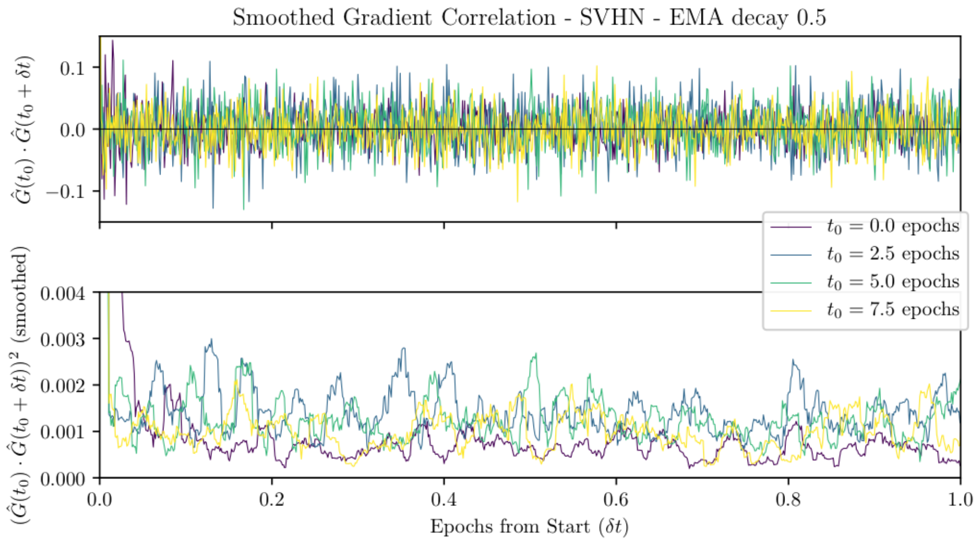


Figure 18: The gradient exhibits rapid, long-lived oscillations over the course of training, even when using adaptive optimizers such as Adam. These oscillations are typical when optimizing functions with a large Hierarchy in the Hessian spectrum. We measure the moving average of the gradient with decay 0.5, computing its correlations over time. Results are shown for a simple CNN trained on SVHN using the Adam optimizer.

## E Comments on Optimization

### E.1 Deterministic Training Performs Poorly

From the discussion of the noise scale in Section 2, we expect that at large batch size $B \gg \mathcal{B}_{\text{noise}}$ we can obtain a very good estimate of the gradient. This would then suggest a minimally stochastic approach to training, where at each step one performs a line search of the true loss in the direction of the true gradient, and updates parameters accordingly.

This nearly deterministic 'greedient descent' method performs poorly in practice [WRLG18]. While its first few steps tend to decrease the loss significantly, subsequent step sizes decrease rapidly and provide minimal further training progress. In fact, when training with a fixed learning rate, we have observed that training often tends towards a regime where the optimal step size (determined by a line search in the direction of the parameter update) is almost exactly half of the actual update magnitude. We have found that this phenomenon occurs regardless of the learning rate (scanning over several orders of magnitude), and seems common to a variety of different models, as shown in Figures 17 and 18. A natural interpretation of these results is that large Hessian directions are dominating the update [GARD18], so that training involves rapid oscillations, as seen in Figure 18 (see [Goh17] for an intuitive picture of these oscillations). Because of this, line searches do not appear to be useful in determining the optimal step size for a full training run.

### E.2 Motivations for Learning Rate Scaling Rules

The learning rate scaling rule from Appendix A.2 can be motivated as follows. Equation A.3 generalizes Equation 2.6, which was derived for plain SGD. The SGD linear scaling rule ($\alpha = 1$) means that the step size per data example stays fixed up to $B_*$; one might intuitively expect that this is necessary to avoid diminishing returns as $B$ increases. In the case of Adam[21], we can use noise scale considerations to motivate the generalization to $0.5 < \alpha < 1$. The Adam update to the parameter $\theta_i$ takes the form

$$\delta\theta_i = \epsilon \frac{\mathbb{E}_{\beta_1}\left[G_i\right]}{\sqrt{\mathbb{E}_{\beta_2}\left[G_i^2\right]} + \epsilon_{\text{Adam}}}$$

where $\mathbb{E}_\beta$ refers to an exponentially-weighted moving average with decay parameter $\beta$, and $G$ refers to a gradient from a batch of size $B$. If we disregard $\beta_1, \beta_2$, and $\epsilon_{\text{Adam}}$, this is roughly equivalent to

$$\delta\theta_i \approx \epsilon \frac{\operatorname{sign}\left(\mathbb{E}\left[G_i\right]\right)}{\sqrt{1 + \frac{s_i}{\mathbb{E}[G_i]^2}}}.$$

where $s_i$ is the variance of $G_i$ over timesteps. If the step-to-step noise in the gradient is primarily due to batch statistics, $s_i$ should scale inversely with $B$. Comparing with 2.6, this implies a square-root scaling rule ($\alpha = 0.5$) to maintain a constant learning rate per data example. However, since $\beta_2$ is often set to large values around 0.999, the second moment accumulator may not have time to adapt to quick changes in the gradient noise; this pushes $\alpha$ back towards 1.0. This may explain the variation in $\alpha$ between different tasks.

### E.3 Preliminary Tests of Generalization

Throughout the paper we have studied the noise scale as a function of the training loss or RL score. But for non-RL tasks, it is also interesting to study the relationship between the noise scale and the critical batch size associated with minimization of the test loss. The difference between the training and test results provides information about generalization. In Figure 19 we report results using the test loss for the small image classification datasets.

As expected, early in training there is no difference between train and test results. However, at the very end of training we observe a small dip in $\mathcal{B}_{\text{crit}}$ for the test loss, which appears to occur consistently across datasets. It would be very interesting to further investigate this phenomenon in the future.

[21]These considerations apply equally well to RMSProp.

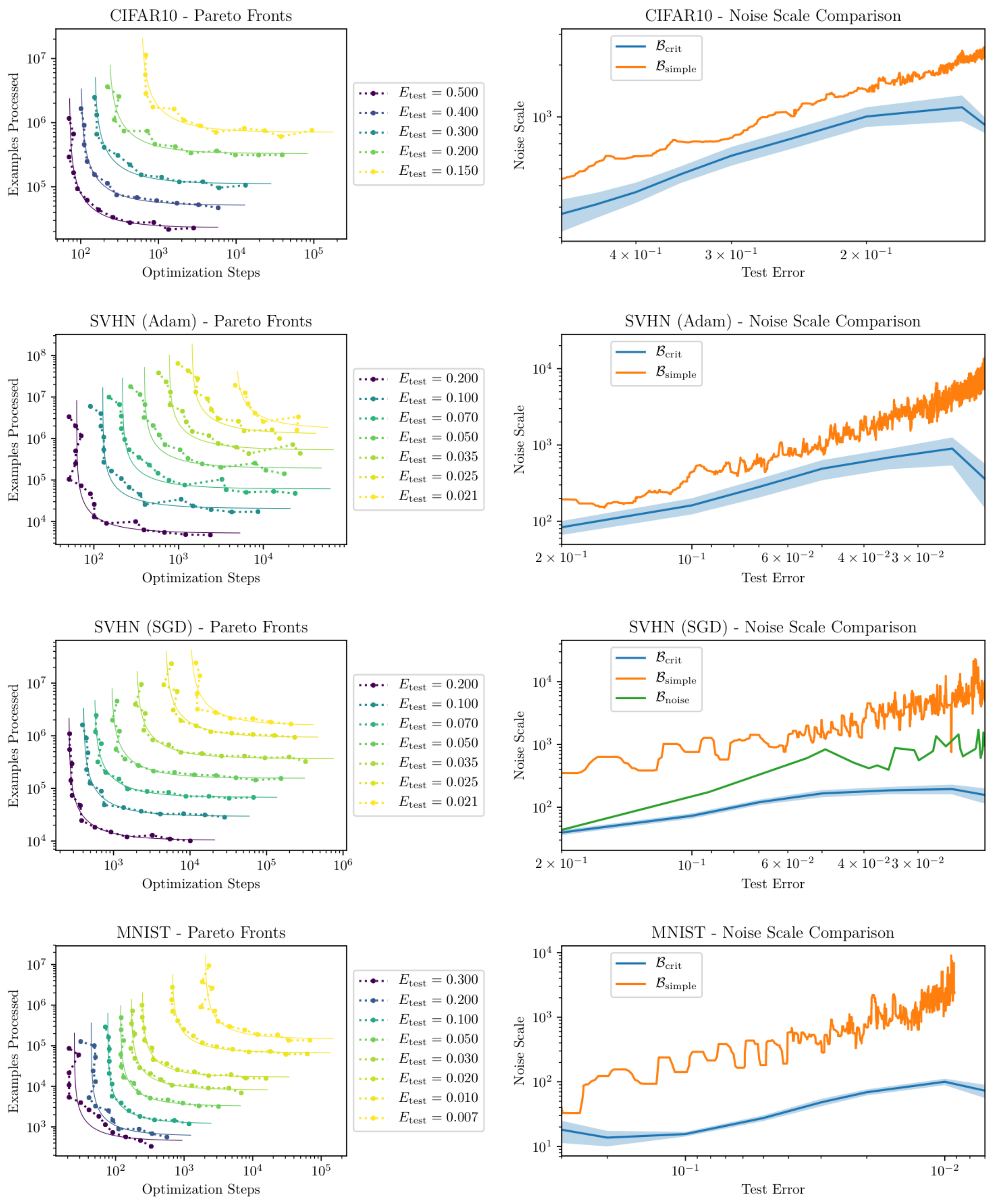


Figure 19: Scaling behavior of image classification tasks, using test set goals rather than Train set goals. These results should be compared to those of Figure 14, which use training goals.

## References

[AAG$^+$18] Igor Adamski, Robert Adamski, Tomasz Grel, Adam Jędrych, Kamil Kaczmarek, and Henryk Michalewski. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes, 2018, 1801.02852. 2, 14

[AH18] Dario Amodei and Danny Hernandez. AI and Compute, May 2018. URL `https://blog.openai.com/ai-and-compute/`. 2

[ASF17] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes, 2017, 1711.04325. 2

[BCD$^+$18] Greg Brockman, Brooke Chan, Przemyslaw Debiak, Christy Dennison, David Farhi, Rafal Józefowicz, Jakub Pachocki, Michael Petrov, Henrique Pondé, Jonathan Raiman, Szymon Sidor, Jie Tang, Filip Wolski, and Susan Zhang. OpenAI Five, Jun 2018. URL `https://blog.openai.com/openai-five/`. 1, 2, 3, 9, 13, 14, 16, 19

[BCN16] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2016, 1606.04838. 14

[BCNW12] Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical Programming*, 134(1):127–155, Aug 2012. doi:10.1007/s10107-012-0572-5. 14

[BCP$^+$16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016, 1606.01540. 19

[BDS18] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2018, 1809.11096. 2

[BGM17] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations, 2017. URL `https://openreview.net/forum?id=SkkTMpjex`. 14

[BNVB12] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, 2012, 1207.4708. 13, 19

[BRH16] Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates, 2016, 1612.05086. 14

[CDH$^+$16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016, 1606.03657. 19

[CMS$^+$13] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2013, 1312.3005. 12, 19

[CWZ$^+$18] Lingjiao Chen, Hongyi Wang, Jinman Zhao, Dimitris Papailiopoulos, and Paraschos Koutris. The effect of network width on the performance of large-batch training, 2018, 1806.03791. 14

[DDS$^+$09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. URL `http://www.image-net.org/papers/imagenet_cvpr09.pdf`. 12, 18

[DHK$^+$17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017. 19

[DNG17] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks, 2017, 1712.02029. 14

[DYJG16] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Big batch sgd: Automated inference using adaptive batch sizes, 2016, 1610.05792. 14

[GARD18] Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. 2018, arXiv:1812.04754. 30

[GDG$^+$17] 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, 2017, 1706.02677. 2, 14, 20

[Goh17] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi:10.23915/distill.00006. 8, 30

[Goo] Google. Tensorflow official models. URL `https://github.com/tensorflow/models/tree/master/official/resnet`. 18

[GVS14] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems, 2014, 1412.6544. 15

[GVY$^+$18] 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, 2018, 1811.12941. 14

[HHS17] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. 2017, 1705.08741. 8, 28

[HMG07] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{TM}}$: A bayesian skill rating system. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 569–576. MIT Press, 2007. URL `http://papers.nips.cc/paper/3079-trueskilltm-a-bayesian-skill-rating-system.pdf`. 19

[HNA$^+$17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017, 1712.00409. 15

[HQB$^+$18] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018, 1803.00933. 2, 14

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. doi:10.1162/neco.1997.9.8.1735. 19

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, 2015, 1512.03385. 12, 18, 20

[IES$^+$18] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Are deep policy gradient algorithms truly policy gradient algorithms?, 2018, 1811.02553. 14

[JSH$^+$18] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018, 1807.11205. 2, 12, 14

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014, 1412.6980. 5, 7, 12, 17, 18, 19

[KMN$^+$16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016, 1609.04836. 8, 15, 28

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`. 12, 18

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013, arXiv:1312.6114. 12, 19

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`. 12, 18

[LFLY18] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes, 2018, 1804.08838. 8

[MBB17] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning, 2017, 1712.06559. 14

[MBM$^+$16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, 2016, 1602.01783. 13, 19

[MHB17] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic gradient descent as approximate bayesian inference. 2017, 1704.04289. 14, 25, 26

[NIS] Selecting sample sizes. URL `https://www.itl.nist.gov/div898/handbook/ppc/section3/ppc333.htm`. 14

[NVL$^+$15] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks, 2015, 1511.06807. 15

[NWC$^+$11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011. URL `http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf`. 12, 18

[OEGA18] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation, 2018, 1806.00187. 2

[Ope17] OpenAI. More on Dota 2, Aug 2017. URL `https://blog.openai.com/more-on-dota-2/`. 22

[PKYC18] Raul Puri, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro. Large scale language modeling: Converging on 40gb of text in four hours, 2018, 1808.01371. 2

[SA18] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning, 2018, 1803.02811. 2, 14, 19

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, 2015, 1508.07909. 19

[SKYL17] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size, 2017, 1711.00489. 2, 14, 25, 27

[SL17] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent, 2017, 1710.06451. 14

[SLA$^+$18] Christopher J. Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training, 2018, arXiv:1811.03600. 8, 13, 14

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1139–1147, 17–19 Jun 2013. URL `http://proceedings.mlr.press/v28/sutskever13.html`. 18, 19

[SWD$^+$17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017, 1707.06347. 13, 19

[SZL13] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 343–351. JMLR.org, 2013. URL `http://jmlr.org/proceedings/papers/v28/schaul13.html`. 14, 26

[SZT17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information, 2017, 1703.00810. 14

[Wik] Sample size determination. URL `https://en.wikipedia.org/wiki/Sample_size_determination#Estimation_of_a_mean`. 14

[WRLG18] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization, 2018, 1803.02021. 8, 14, 29

[YGG17] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017, 1708.03888. 2, 12, 14

[YPL$^+$17] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning, 2017, 1706.05699. 14, 15

[YZH$^+$17] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes, 2017, 1709.05011. 2, 14