Measuring the Effects of Data Parallelism on Neural Network Training

Christopher J. Shallue[*]   SHALLUE@GOOGLE.COM

Jaehoon Lee[*][†]   JAEHLEE@GOOGLE.COM

Joseph Antognini[†]   JOE.ANTOGNINI@GMAIL.COM

Jascha Sohl-Dickstein   JASCHASD@GOOGLE.COM

Roy Frostig   FROSTIG@GOOGLE.COM

George E. Dahl   GDAHL@GOOGLE.COM

Google Brain
1600 Amphiteatre Parkway
Mountain View, CA, 94043, USA

Editor: Rob Fergus

Abstract

Recent hardware developments have dramatically increased the scale of data parallelism available for neural network training. Among the simplest ways to harness next-generation hardware is to increase the batch size in standard mini-batch neural network training algorithms. In this work, we aim to experimentally characterize the effects of increasing the batch size on training time, as measured by the number of steps necessary to reach a goal out-of-sample error. We study how this relationship varies with the training algorithm, model, and data set, and find extremely large variation between workloads. Along the way, we show that disagreements in the literature on how batch size affects model quality can largely be explained by differences in metaparameter tuning and compute budgets at different batch sizes. We find no evidence that larger batch sizes degrade out-of-sample performance. Finally, we discuss the implications of our results on efforts to train neural networks much faster in the future. Our experimental data is publicly available as a database of 71,638,836 loss measurements taken over the course of training for 168,160 individual models across 35 workloads.

Keywords: neural networks, stochastic gradient descent, data parallelism, batch size, deep learning

1. Introduction

Neural networks have become highly effective at a wide variety of prediction tasks, including image classification, machine translation, and speech recognition. The dramatic improvements in predictive performance over the past decade have partly been driven by advances in hardware for neural network training, which have enabled larger models to be trained on larger datasets than ever before. However, although modern GPUs and custom

---

*. Both authors contributed equally.

†. Work done as a member of the Google AI Residency program (g.co/airesidency).

accelerators have made training neural networks orders of magnitude faster, training time still limits both the predictive performance of these techniques and how widely they can be applied. For many important problems, the best models are still improving at the end of training because practitioners cannot afford to wait until the performance saturates. In extreme cases, training must end before completing a single pass over the data (e.g. Anil et al., 2018). Techniques that speed up neural network training can significantly benefit many important application areas. Faster training can facilitate dramatic improvements in model quality by allowing practitioners to train on more data (Hestness et al., 2017), and by decreasing the experiment iteration time, allowing researchers to try new ideas and configurations more rapidly. Faster training can also allow neural networks to be deployed in settings where models have to be updated frequently, for instance when new models have to be produced when training data get added or removed.

Data parallelism is a straightforward and popular way to accelerate neural network training. For our purposes, data parallelism refers to distributing training examples across multiple processors to compute gradient updates (or higher-order derivative information) and then aggregating these locally computed updates. As long as the training objective decomposes into a sum over training examples, data parallelism is model-agnostic and applicable to any neural network architecture. In contrast, the maximum degree of *model parallelism* (distributing parameters and computation across different processors for the same training examples) depends on the model size and structure. Although data parallelism can be simpler to implement, ultimately, large scale systems should consider all types of parallelism at their disposal. In this work, we focus on the costs and benefits of data parallelism in the synchronous training setting.

Hardware development is trending towards increasing capacity for data parallelism in neural network training. Specialized systems using GPUs or custom ASICs (e.g. Jouppi et al., 2017) combined with high-performance interconnect technology are unlocking unprecedented scales of data parallelism where the costs and benefits have not yet been well studied. On the one hand, if data parallelism can provide a significant speedup at the limits of today's systems, we should build much bigger systems. On the other hand, if additional data parallelism comes with minimal benefits or significant costs, we might consider designing systems to maximize serial execution speed, exploit other types of parallelism, or even prioritize separate design goals such as power use or cost.

There is considerable debate in the literature about the costs and benefits of data parallelism in neural network training and several papers take seemingly contradictory positions. Some authors contend that large-scale data parallelism is harmful in a variety of ways, while others contend that it is beneficial. The range of conjectures, suggestive empirical results, and folk knowledge seems to cover most of the available hypothesis space. Answering these questions definitively has only recently become important (as increasing amounts of data parallelism have become practical), so it is perhaps unsurprising that the literature remains equivocal, especially in the absence of sufficiently comprehensive experimental data.

In this work, we attempt to provide the most rigorous and extensive experimental study on the effects of data parallelism on neural network training to date. In order to achieve this goal, we consider realistic workloads up to the current limits of data parallelism. We try to avoid making assumptions about how the optimal metaparameters vary as a function of batch size. Finally, in order to guide future work, we consider any remaining limitations in

our methodology, and we discuss what we see as the most interesting unanswered questions that arise from our experiments.

1.1. Scope

We restrict our attention to variants of mini-batch stochastic gradient descent (SGD), which are the dominant algorithms for training neural networks. These algorithms iteratively update the model's parameters using an estimate of the gradient of the training objective. The gradient is estimated at each step using a different subset, or *(mini-) batch*, of training examples. See Section 2.2 for a more detailed description of these algorithms. A data-parallel implementation computes gradients for different training examples in each batch in parallel, and so, in the context of mini-batch SGD and its variants, we equate the batch size with the amount of data parallelism.[1] We restrict our attention to synchronous SGD because of its popularity and advantages over asynchronous SGD (Chen et al., 2016).

Practitioners are primarily concerned with out-of-sample error and the cost they pay to achieve that error. Cost can be measured in a variety of ways, including training time and hardware costs. Training time can be decomposed into number of steps multiplied by average time per step, and hardware cost into number of steps multiplied by average hardware cost per step. The per-step time and hardware costs depend on the practitioner's hardware, but the number of training steps is hardware-agnostic and can be used to compute the total costs for any hardware given its per-step costs. Furthermore, in an idealized data-parallel system where the communication overhead between processors is negligible, training time depends only on the number of training steps (and not the batch size) because the time per step is independent of the number of examples processed. Indeed, this scenario is realistic today in systems like TPU pods[2], where there are a range of batch sizes for which the time per step is almost constant. Since we are primarily concerned with training time, we focus on number of training steps as our main measure of training cost.

An alternative hardware-agnostic measure of training cost is the number of training examples processed, or equivalently the number of passes (*epochs*) over the training data. This measure is suitable when the per-step costs are proportional to the number of examples processed (e.g. hardware costs proportional to the number of floating point operations). However, the number of epochs is not a suitable measure of training time in a data-parallel system—it is possible to reduce training time by using a larger batch size and processing *more* epochs of training data, provided the number of training steps decreases.

In light of practitioners' primary concerns of out-of-sample error and the resources needed to achieve it, we believe the following questions are the most important to study to understand the costs and benefits of data parallelism with mini-batch SGD and its variants:

1. What is the relationship between batch size and number of training steps to reach a goal out-of-sample error?

2. What governs this relationship?

3. Do large batch sizes incur a cost in out-of-sample error?

---

1. Mini-batch SGD can be implemented in a variety of ways, including data-serially, but a data-parallel implementation is always possible given appropriate hardware.

2. `https://www.blog.google/products/google-cloud/google-cloud-offer-tpus-machine-learning/`.

1.2. Contributions of This Work

1. We show that the relationship between batch size and number of training steps to reach a goal out-of-sample error has the same characteristic form across six different families of neural network, three training algorithms, and seven data sets.

   Specifically, for each workload (model, training algorithm, and data set), increasing the batch size initially decreases the required number of training steps proportionally, but eventually there are diminishing returns until finally increasing the batch size no longer changes the required number of training steps. To the best of our knowledge, we are the first to experimentally validate this relationship across models, training algorithms, and data sets while independently tuning the learning rate, momentum, and learning rate schedule (where applicable) for each batch size. Unlike prior work that made strong assumptions about these metaparameters, our results reveal a universal relationship that holds across all workloads we considered, across different error goals, and when considering either training error or out-of-sample error.

2. We show that the maximum useful batch size varies significantly between workloads and depends on properties of the model, training algorithm, and data set. Specifically, we show that:

   (a) SGD with momentum (as well as Nesterov momentum) can make use of much larger batch sizes than plain SGD, suggesting future work to study the batch size scaling properties of other algorithms.

   (b) Some models allow training to scale to much larger batch sizes than others. We include experimental data on the relationship between various model properties and the maximum useful batch size, demonstrating that the relationship is not as simple as one might hope from previous work (e.g. wider models do *not* always scale better to larger batch sizes).

   (c) The effect of the data set on the maximum useful batch size tends to be smaller than the effects of the model and training algorithm, and does not depend on data set size in a consistent way.

3. We show that the optimal values of training metaparameters do not consistently follow any simple relationships with the batch size. In particular, popular learning rate heuristics—such as linearly scaling the learning rate with the batch size— do not hold across all problems or across all batch sizes.

4. Finally, by reviewing the specifics of the experimental protocols used in prior work, we at least partially reconcile conflicting stances in the literature on whether increasing the batch size degrades model quality. Specifically, we show that assumptions about computational budgets and the procedures for selecting metaparameters at different batch sizes can explain many of the disagreements in the literature. We find no evidence that increasing the batch size necessarily degrades model quality, but additional regularization techniques may become important at larger batch sizes.

1.3. Experimental Data

We release our raw experimental data for any further analysis by the research community.[3] Our database contains 454 combinations of workload (model, data set, training algorithm) and batch size, each of which is associated with a metaparameter search space and a set of models trained with different configurations sampled from the search space. In total, our data contains 71,638,836 loss measurements taken over the course of training for 168,160 individual models. Together, these measurements make up the training curves of all of the individual models we trained, and can be used to reproduce all plots in this paper.[4]

2. Setup and Background

In this section we set up the basic definitions and background concepts used throughout the paper.

2.1. Learning

A *data distribution* is a probability distribution $\mathcal{D}$ over a data domain $\mathcal{Z}$. For example, we might consider a supervised learning task over a domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the set of 32-by-32-pixel color images and $\mathcal{Y}$ is the set of possible labels denoting what appears in the image. A *training set* $z_1, \ldots, z_n \in \mathcal{Z}$ is a collection of *examples* from the data domain, conventionally assumed to be drawn i.i.d. from the data distribution $\mathcal{D}$.

A machine learning *model* is a function that, given *parameters* θ from some set $\Theta \subset \mathbb{R}^d$, and given a data point $z \in \mathcal{Z}$, produces a prediction whose quality is measured by a differentiable non-negative scalar-valued loss function.[5] We denote by $\ell(\theta; z)$ the loss of a prediction made by the model, under parameters θ, on the data point z. We denote by L the *out-of-sample loss* or *expected loss*:

$$L(\theta) = \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[\ell(\theta; z) \right], \tag{1}$$

and by $\hat{L}$ the *empirical average loss* under a data set $S = (z_1, \ldots, z_n)$:

$$\hat{L}(\theta; S) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; z_i). \tag{2}$$

When S is the training set, we call $\hat{L}$ the *average training loss*. We will say that the data source $\mathcal{D}$, loss ℓ, and model with parameter set Θ together specify a learning *task*, in which our aim is to find parameters θ that achieve low out-of-sample loss (Equation 1), while given access only to n training examples. A common approach is to find parameters of low average training loss (Equation 2) as an estimate of the out-of-sample loss (Shalev-Shwartz and Ben-David, 2014).

When minimizing average training loss $\hat{L}$, it is common to add regularization penalties to the objective function. For a differentiable penalty $R : \Theta \to \mathbb{R}_+$, regularization weight

---

3. `https://github.com/google-research/google-research/tree/master/batch_science`
4. `https://colab.research.google.com/github/google-research/google-research/blob/master/batch_science/reproduce_paper_plots.ipynb`
5. Technically, the loss need only be sub-differentiable. Extending our setup to this end is straightforward.

$\lambda > 0$, and training set S, the training objective might be

$$J(\theta) = \hat{L}(\theta; S) + \lambda R(\theta). \tag{3}$$

In practice, we often approach a task by replacing its loss with another that is more amenable to training. For instance, in supervised classification, we might be tasked with learning under the 0/1 loss, which is an indicator of whether a prediction is correct (e.g. matches a ground-truth label), but we train by considering instead a surrogate loss (e.g. the logistic loss) that is more amenable to continuous optimization. When the surrogate loss bounds the original, achieving low loss under the surrogate implies low loss under the original. To distinguish the two, we say *error* to describe the original loss (e.g. 0/1), and we save *loss* to refer to the surrogate used in training.

2.2. Algorithms

The dominant algorithms for training neural networks are based on *mini-batch stochastic gradient descent* (SGD, Robbins and Monro, 1951; Kiefer et al., 1952; Rumelhart et al., 1986; Bottou and Bousquet, 2008; LeCun et al., 2015). Given an initial point $\theta_0 \in \Theta$, mini-batch SGD attempts to decrease the objective J via the sequence of iterates

$$\theta_t \leftarrow \theta_{t-1} - \eta_t g(\theta_{t-1}; B_t),$$

where each B_t is a random subset of training examples, the sequence $\{\eta_t\}$ of positive scalars is called the *learning rate*, and where, for any $\theta \in \Theta$ and $B \subset S$,

$$g(\theta; B) = \frac{1}{|B|} \sum_{z \in B} \nabla \ell(\theta; z) + \lambda \nabla R(\theta). \tag{4}$$

When the examples B are a uniformly random subset of training examples, $g(\theta; B)$ forms an unbiased estimate of the gradient of the objective J that we call a *stochastic gradient*. In our larger-scale experiments, when we sample subsequent batches B_t, we actually follow the common practice of cycling through permutations of the training set (Shamir, 2016). The result of mini-batch SGD can be any of the iterates θ_t for which we estimate that $L(\theta_t)$ is low using a validation data set.

Variants of SGD commonly used with neural networks include SGD with momentum (Polyak, 1964; Rumelhart et al., 1986; Sutskever et al., 2013), Nesterov momentum (Nesterov, 1983; Sutskever et al., 2013), RMSProp (Hinton et al., 2012), and Adam (Kingma and Ba, 2015). All of these optimization procedures, or *optimizers*, interact with the training examples only by repeatedly computing stochastic gradients (Equation 4), so they support the same notion of batch size that we equate with the scale of data parallelism. In this work, we focus on the SGD, SGD with momentum, and Nesterov momentum optimizers. The latter two optimizers are configured by a learning rate $\{\eta_t\}$ and a scalar $\gamma \in (0, 1)$ that we call *momentum*. They define the iterates[6]

SGD with momentum	Nesterov momentum
$v_{t+1} \leftarrow \gamma v_t + g(\theta_t; B_t)$	$v_{t+1} \leftarrow \gamma v_t + g(\theta_t; B_t)$
$\theta_{t+1} \leftarrow \theta_t - \eta_t v_{t+1}$	$\theta_{t+1} \leftarrow \theta_t - \eta_t g(\theta_t; B_t) - \eta_t \gamma v_{t+1},$

---

6. These rules take slightly different forms across the literature and across library implementations. We present and use the update rules from the `MomentumOptimizer` class in TensorFlow (Abadi et al., 2016).

given $v_0 = 0$ and an initial θ_0. Note that plain SGD can be recovered from either optimizer by taking $\gamma = 0$. The outcome of using these optimizers should therefore be no worse if than SGD, in any experiment, the momentum γ is tuned across values including zero.

If we run SGD with momentum under a constant learning rate $\eta_t = \eta$, then, at a given iteration t, the algorithm computes

$$\theta_{t+1} = \theta_t - \eta v_{t+1} = \theta_0 - \eta \sum_{u=0}^{t} v_{u+1} = \theta_0 - \eta \sum_{u=0}^{t} \sum_{s=0}^{u} \gamma^{u-s} g(\theta_s; B_s).$$

For any fixed $\tau \in \{0, \ldots, t\}$, the coefficient accompanying the stochastic gradient $g(\theta_\tau; B_\tau)$ in the above update is $\eta \sum_{u=\tau}^{t} \gamma^{u-\tau}$. We define the *effective learning rate*, η^{eff} as the value of this coefficient at the end of training ($t = T$), in the limit of a large number of training steps ($T \to \infty$, while τ is held fixed):

$$\eta^{\text{eff}} = \lim_{T \to \infty} \sum_{u=\tau}^{T} \eta \gamma^{u-\tau} = \frac{\eta}{1 - \gamma}.$$

Put intuitively, η^{eff} captures the contribution of a given mini-batch gradient to the parameter values at the end of training.

2.3. Additional Terminology in Experiments

A *data-parallel implementation* of mini-batch SGD (or one of its variants) computes the summands of Equation 4 in parallel and then synchronizes to coordinate their summation.

The models and algorithms in our experiments are modifiable by what we call *meta-parameters*.[7] These include architectural choices, such as the number of layers in a neural network, and training parameters, such as learning rates $\{\eta_t\}$ and regularization weights λ. When we use the term *model*, we typically assume that all architectural metaparameters have been set. In our experiments, we *tune* the training metaparameters by selecting the values that yield the best performance on a validation set. We use the term *workload* to jointly refer to a data set, model, and training algorithm.

3. Related Work

In this section we review prior work related to our three main questions from Section 1.1. First we review studies that considered the relationship between batch size and number of training steps (Questions 1 and 2), and then we review studies that considered the effects of batch size on solution quality (Question 3).

3.1. Steps to Reach a Desired Out-Of-Sample Error

We broadly categorize the related work on this topic as either analytical or empirical in nature.

---

7. Sometimes called "hyperparameters," but we prefer a different name so as not to clash with the notion of hyperparameters in Bayesian statistics.

3.1.1. ANALYTICAL STUDIES

Convergence upper bounds from the theory of stochastic (convex) optimization can be specialized to involve terms dependent on batch size, so in this sense they comprise basic related work. These upper bounds arise from worst-case analysis, and moreover make convexity and regularity assumptions that are technically violated in neural network training, so whether they predict the actual observed behavior of our experimental workloads is an empirical question in its own right.

Given a sequence of examples drawn i.i.d. from a data source, an upper bound on the performance of SGD applied to L-Lipschitz convex losses is (Hazan, 2016; Shalev-Shwartz and Ben-David, 2014)

$$J(\theta_T) - J^\star \leq O\left(\sqrt{\frac{L^2}{T}}\right), \tag{5}$$

for any batch size. Here, J is the objective function, $J^\star$ is its value at the global optimum, and θ_T denotes the final output of the algorithm supposing it took T iterations.[8] Meanwhile, when losses are convex and the objective is H-smooth, accelerated parallel mini-batch SGD enjoys the bound (Lan, 2012)

$$J(\theta_T) - J^\star \leq O\left(\frac{H}{T^2} + \sqrt{\frac{L^2}{Tb}}\right), \tag{6}$$

where b is the batch size.

Compared to sequential processing without batching (i.e. a batch size of one), the bounds Equation 5 and Equation 6 offer two extremes, respectively:

1. **No benefit:** Increasing the batch size b does not change the number of steps to convergence, as per Equation 5.

2. **A b-fold benefit:** The term in Equation 6 proportional to $1/\sqrt{Tb}$ dominates the bound. Increasing the batch size b by a multiplicative factor decreases the number of steps T to a given achievable objective value by the same factor.

In other words, under these simplifications, batching cannot hurt the asymptotic guarantees of steps to convergence, but it could be wasteful of examples. The two extremes imply radically different guidance for practitioners, so the critical task of establishing a relationship between batch size and number of training steps remains one to resolve experimentally.

A few recent papers proposed analytical notions of a critical batch size: a point at which a transition occurs from a b-fold benefit to no benefit. Under assumptions including convexity, Ma et al. (2018) derived such a critical batch size, and argued that a batch size of one is optimal for minimizing the number of training epochs required to reach a given target error. Under different assumptions, Yin et al. (2018) established a critical batch size and a pathological loss function that together exhibit a transition from a b-fold benefit to no benefit. Although they ran experiments with neural networks, their experiments were designed to investigate the effect of data redundancy and do not provide enough

---

8. Not necessarily the Tth iterate, which may differ from θ_T if the algorithm averages its iterates.

information to reveal the empirical relationship between batch size and number of training steps. Focusing on linear least-squares regression, Jain et al. (2018) also derived a threshold batch size in terms of the operator norm of the objective's Hessian and a constant from a fourth-moment bound on example inputs.

To our knowledge, in all previous work that analytically derived a critical batch size, the thresholds defined are either (i) parameter-dependent, or (ii) specific to linear least-squares regression. A critical batch size that depends on model parameters can change over the course of optimization; it is not a problem-wide threshold that can be estimated efficiently *a priori*. Focusing on least-squares has issues as well: while it sheds intuitive light on how batching affects stochastic optimization locally, the quantities defined inherently cannot generalize to the non-linear optimization setting of neural network training, both because the objective's Hessian is not constant across the space of parameters as it is in a quadratic problem, and more broadly because it is unclear whether the Hessian of the objective is still the correct analogue to consider.

3.1.2. EMPIRICAL STUDIES

Wilson and Martinez (2003) investigated the relationship between batch size and training speed for plain mini-batch SGD. They found that a simple fully connected neural network took more epochs to converge with larger batch sizes on a data set of 20,000 examples, and also that using a batch size equal to the size of the training set took more epochs to converge than a batch size of one on several small data sets of size ≤ 600. However, their experimental protocol and assumptions limit the conclusions we can draw from their results. One issue is that training time was measured to different out-of-sample errors for different batch sizes on the same data set. To compare training speed fairly, the error goal should be fixed across all training runs being compared. Additionally, only four learning rates were tried for each data set, but quite often the best learning rate was at one of the two extremes and it appeared that a better learning rate might be found outside of the four possibilities allowed. Finally, despite the contention of the authors, their results do not imply slower training with larger batch sizes in data-parallel training: for the most part, their larger batch size experiments took fewer training steps than the corresponding batch size one experiments.

In the last few years, increasingly specialized computing systems have spurred practitioners to try much larger batch sizes than ever before, while increasingly promising results have driven hardware designers to create systems capable of even more data parallelism. Chen et al. (2016) used a pool of synchronized worker machines to increase the effective batch size of mini-batch SGD. They demonstrated speedups in both wall time and steps to convergence for an Inception model (Szegedy et al., 2016) on ImageNet (Russakovsky et al., 2015) by scaling the effective batch size from 1,600 to 6,400. More recently, Goyal et al. (2017) showed that the number of training epochs could be held constant across a range of batch sizes to achieve the same validation error for ResNet-50 (He et al., 2016a) on ImageNet. Holding the number of training epochs constant is equivalent to scaling the number of training steps inversely with the batch size, and this reduction in training steps with increasing batch size produced nearly proportional wall time speedups on their hardware. Although this hints at a *b*-fold benefit regime in which increasing the batch size reduces the

number of training steps by the same factor, the authors did not attempt to minimize the number of training steps (or epochs) required to reach the goal at each batch size separately. It is unclear whether any of the batch sizes that achieved the goal could do so in fewer steps than given, or how many steps the other batch sizes would have needed to achieve the same error goal.

Two studies performed concurrently with this work also investigated the relationship between batch size and training speed for neural networks. Chen et al. (2018) provide experimental evidence of a problem-dependent critical batch size after which a b-fold benefit is no longer achieved for plain mini-batch SGD. They contend that wider and shallower networks have larger critical batch sizes, and while their empirical results are equivocal for this particular claim, they show that the threshold batch size can depend on aspects of both the data set and the model. Additionally, Golmant et al. (2018) studied how three previously proposed heuristics for adjusting the learning rate as a function of batch size (linear scaling, square root scaling, and no scaling) affect the number of training steps required to reach a particular result. They found that if the learning rate is tuned for the the smallest batch size only, all three of these common scaling techniques break down for larger batch sizes and result in either (i) divergent training, or (ii) training that cannot reach the error goal within a fixed number of training epochs. They also describe a basic relationship between batch size and training steps to a fixed error goal, which is comprised of three regions: b-fold benefit initially, then diminishing returns, and finally no benefit for all batch sizes greater than a maximum useful batch size. However, their results are inconclusive because (i) not all model and data set pairs exhibit this basic relationship, (ii) it does not appear consistently across error goals, and (iii) the relationship is primarily evident in training error but not out-of-sample error. These inconsistent results may be due to suboptimal pre-determined learning rates arising from the scaling rules, especially at larger batch sizes. Finally, they also found that the maximum useful batch size depends on aspects of the model and the data set type, but not on the data set size. Since all their experiments use plain mini-batch SGD, their results are unable to reveal any effects from the choice of optimizer and might not generalize to other popular optimizers, such as SGD with momentum.

3.2. Solution Quality

The literature contains some seemingly conflicting claims about the effects of batch size on solution quality (out-of-sample error at the conclusion of training). Primarily, the debate centers on whether increasing the batch size incurs a cost in solution quality. Keskar et al. (2017) argue that large batch[9] training converges to so-called "sharp" minima with worse generalization properties. However, Dinh et al. (2017) show that a minimum with favorable generalization properties can be made, through reparameterization, arbitrarily sharp in the same sense. Le Cun et al. (1998) suggest that a batch size of one can result in better solutions because the noisier updates allow for the possibility of escaping from local minima in a descent algorithm. However, they also note that we usually stop training long before

---

9. The term "large batch" is inherently ambiguous, and in this case accompanies experiments in Keskar et al. (2017) that only compare two absolute batch sizes per data set, rather than charting out a curve to its apparent extremes.

reaching any sort of critical point. Hoffer et al. (2017) argue that increasing the batch size need not degrade out-of-sample error at all, assuming training has gone on long enough. Goyal et al. (2017), among others, tested batch sizes larger than those used in Keskar et al. (2017) without noticing any reduction in solution quality. Still, their results with yet larger batch sizes do not rule out the existence of a more sudden degradation once the batch size is large enough. Meanwhile, Goodfellow et al. (2016) state that small batches can provide a regularization effect such that they result in the best observed out-of-sample error, although in this case other regularization techniques might serve equally well.

Alas, the best possible out-of-sample error for a particular model and data set cannot be measured unconditionally due to practical limits on wall time and hardware resources, as well as practical limits on our ability to tune optimization metaparameters (e.g. the learning rate). An empirical study can only hope to measure solution quality subject to the budgets allowed for each model experiment, potentially with caveats due to limitations of the specific procedures for selecting the metaparameters. To the best of our knowledge, all published results handle the training budget issue in exactly one of three ways: by ignoring budgets (train to convergence, which is not always possible); by using a step budget (restrict the number of gradient descent updates performed); or by using an epoch budget (restrict number of training examples processed).[10] Furthermore, while some published results tune the learning rate anew for each batch size, others tune for only a single batch size and use a preordained heuristic to set the learning rate for the remaining batch sizes (the most common heuristics are constant, square root, and linear learning rate scaling rules). Tuning metaparameters at a single batch size and then heuristically adjusting them for others could clearly create a systematic advantage for trials at batch sizes near to the one tuned. All in all, the conclusions we can draw from previous studies depend on the budgets they assume and on how they select metaparameters across batch sizes. The following subsections attempt an investigation of their experimental procedures to this end.

3.2.1. Studies That Ignore Budgets

All studies in this section compared solution quality for different batch sizes after deeming their models to have converged. They determined training stopping time by using either manual inspection, convergence heuristics, or fixed compute budgets that they considered large enough to guarantee convergence.[11]

Keskar et al. (2017) trained several neural network architectures on MNIST and CIFAR-10, each with two batch sizes, using the Adam optimizer and without changing the learning rate between batch sizes. They found that the larger batch size consistently achieved worse out-of-sample error after training error had ceased to improve. However, all models used batch normalization (Ioffe and Szegedy, 2015) and presumably computed the batch nor-

---

10. Of course, there are budgets in between an epoch budget and a step budget that might allow the possibility of trading off time, computation, and/or solution quality. For example, it may be possible to increase the number of training epochs and still take fewer steps to reach the same quality solution. However, we are not aware of work that emphasizes these budgets.

11. As discussed further in Section 4.8, we find that millions of training steps for small batch sizes, or thousands of epochs for large batch sizes, are required to saturate performance even for data sets as small and simple as MNIST. In our experiments, this corresponded to more than 25 hours of wall-time for each metaparameter configuration.

malization statistics using the full batch size. For a fair comparison between batch sizes, batch normalization statistics should be computed over the same number of examples or else the training objective differs between batch sizes (Goyal et al., 2017). Indeed, Hoffer et al. (2017) found that computing batch normalization statistics over larger batches can degrade solution quality, which suggests an alternative explanation for the results of Keskar et al. (2017). Moreover, Keskar et al. (2017) reported that data augmentation eliminated the difference in solution quality between small and large batch experiments.

Smith and Le (2018) trained a small neural network on just 1,000 examples sampled from MNIST with two different batch sizes, using SGD with momentum and without changing the learning rate between batch sizes. They observed that the larger batch size overfit more than the small batch size resulting in worse out-of-sample error, but this gap was mitigated by applying L_2 regularization (Smith and Le, 2018, figures 3 and 8). They also compared a wider range of batch sizes in experiments that either (i) used a step budget without changing the learning rate for each batch size (Smith and Le, 2018, figures 4 and 6), or (ii) varied the learning rate and used a step budget that was a function of the learning rate (Smith and Le, 2018, figure 5). Instead, we focus on the case where the learning rate and batch size are chosen independently.

Breuel (2015a,b) trained a variety of neural network architectures on MNIST with a range of batch sizes, using the SGD and SGD with momentum optimizers with a range of learning rates and momentum values. They found that batch size had no effect on solution quality for LSTM networks (Breuel, 2015a), but found that larger batch sizes achieved worse solutions for fully connected and convolutional networks, and that the scale of the effect depended on the activation function in the hidden and output layers (Breuel, 2015b).

Finally, Chen et al. (2016) observed no difference in solution quality when scaling the batch size from 1,600 to 6,400 for an Inception model on ImageNet when using the RMSProp optimizer and a heuristic to set the learning rate for each batch size.

3.2.2. Studies with Step Budgets

Hoffer et al. (2017) trained neural networks with two different batch sizes on several image data sets. They found that, by computing batch normalization statistics over a fixed number of examples per iteration ("ghost batch normalization"), and by scaling the learning rate with the square root of the batch size instead of some other heuristic, the solution quality arising from the larger batch size was as good as or better than the smaller batch size. However, the largest batch size used was 4,096, which does not rule out an effect appearing at still larger batch sizes, as suggested by the work of Goyal et al. (2017). Moreover, it remains open whether their proposed learning rate heuristic extends to arbitrarily large batch sizes, or whether it eventually breaks down for batch sizes sufficiently far from the base batch size.

3.2.3. Studies with Epoch Budgets

An epoch budget corresponds to fixing the total number of per-example gradient computations, but, in an idealized data-parallel implementation of SGD, it also corresponds to a step (or even wall time) budget that scales inversely with the batch size. With an epoch budget, a larger batch size can only achieve the same solution quality as a smaller batch

size if it achieves perfect scaling efficiency (a b-fold reduction in steps from increasing the batch size, as described in Section 3.1.1).

Masters and Luschi (2018) show that after a critical batch size depending on the model and data set, solution quality degrades with increasing batch size *when using a fixed epoch budget*. Their results effectively show a limited region of b-fold benefit for those model and data set pairs when trained with SGD, although they did not investigate whether this critical batch size depends on the optimizer used, and they did not consider more than one epoch budget for each problem. We reproduced a subset of their experiments and discuss them in Section 5.

Goyal et al. (2017) recently popularized a linear learning rate scaling heuristic for training the ResNet-50 model using different batch sizes. Using this heuristic, a 90 epoch budget, and SGD with momentum without adjusting or tuning the momentum, they increased the batch size from 64 to 8,192 with no loss in accuracy. However, their learning rate heuristic broke down for even larger batch sizes. Inspired by these results, a sequence of follow-up studies applied additional techniques to further increase the batch size while still achieving the same accuracy and using the same 90 epoch budget. These follow-on studies (Codreanu et al., 2017; You et al., 2017; Akiba et al., 2017) confirm that the best solution quality for a given batch size will also depend on the exact optimization techniques used.

There are several additional papers (Lin et al., 2018; Devarakonda et al., 2017; Golmant et al., 2018) with experiments relevant to solution quality that used an epoch budget, tuned the learning rate for the smallest batch size, and then used a heuristic to choose the learning rate for all larger batch sizes. For instance, Devarakonda et al. (2017) and Lin et al. (2018) used linear learning rate scaling and Golmant et al. (2018) tried constant, square root, and linear learning rate scaling heuristics. All of them concluded that small batch sizes have superior solution quality to large batch sizes with a fixed epoch budget, for various notions of "small" and "large." This could just as easily be an artifact of the learning rate heuristics, and a possible alternative conclusion is that these heuristics are limited (as heuristics often are).

4. Experiments and Results

The primary quantity we measure is the number of steps needed to first reach a desired out-of-sample error, or *steps to result*. To measure steps to result, we used seven image and text data sets with training set sizes ranging from 45,000 to 26 billion examples. Table 1 summarizes these data sets and Appendix A provides the full details. We chose six families of neural network to train on these data sets. For MNIST and Fashion MNIST, we chose a simple fully connected neural network and a simple convolutional neural network (CNN). For CIFAR-10, we chose the ResNet-8 model without batch normalization, partly to compare our results to Masters and Luschi (2018), and partly to have a version of ResNet without batch normalization. For ImageNet, we chose ResNet-50, which uses batch normalization and residual connections, and VGG-11, which uses neither. For Open Images, we chose ResNet-50. For LM1B, we chose the Transformer model and an LSTM model. For Common Crawl, we chose the Transformer model. Table 2 summarizes these models and Appendix B provides the full details.

Data Set	Type	Task	Size	Evaluation Metric
MNIST	Image	Classification	55,000	Classification error
Fashion MNIST	Image	Classification	55,000	Classification error
CIFAR-10	Image	Classification	45,000	Classification error
ImageNet	Image	Classification	1,281,167	Classification error
Open Images	Image	Classification (multi-label)	4,526,492	Average precision
LM1B	Text	Language modeling	30,301,028	Cross entropy error
Common Crawl	Text	Language modeling	∼25.8 billion	Cross entropy error

Table 1: Summary of data sets. Size refers to the number of examples in the training set, which we measure in sentences for text data sets. See Appendix A for full details.

Model Class	Sizes	Optimizers	Data Sets	Learning rate schedule
Fully Connected	Various	SGD	MNIST	Constant
Simple CNN	Base Narrow Wide	SGD Momentum Nesterov mom.	MNIST Fashion MNIST	Constant
ResNet	ResNet-8	SGD Nesterov mom.	CIFAR-10	Linear decay
	ResNet-50	Nesterov mom.	ImageNet Open Images	Linear decay
VGG	VGG-11	Nesterov mom.	ImageNet	Linear decay
Transformer	Base Narrow and shallow Shallow Wide	SGD Momentum Nesterov mom.	LM1B Common crawl	Constant
LSTM	—	Nesterov mom.	LM1B	Constant

Table 2: Summary of models. See Appendix B for full details.

Measuring steps to result requires a particular value of out-of-sample error to be chosen as the goal. Ideally, we would select the best achievable error for each task and model, but since validation error is noisy, the best error is sometimes obtained unreliably. Moreover, for some workloads, the validation error continues to improve steadily beyond the maximum practical training time. Therefore, we generally tried to select the best validation error that we could achieve reliably within a practical training time.

Table 2 also shows the learning rate schedule we used for each model and data set. Learning rate schedules are often used to accelerate neural network training, but finding the best schedule is an optimization problem in its own right (Wu et al., 2018). Instead, researchers typically choose from a range of common learning rate functions based on validation performance and individual preference. While most schedules decay the learning rate monotonically over training, some researchers also "warm-up" the learning rate at the start of training (e.g. He et al., 2016a), particularly when training with large batch sizes (Goyal et al., 2017). We ran experiments with both constant learning rates and with learning rate decay. We used decay for ResNet-8, ResNet-50, and VGG-11, which significantly reduced

training time for those models. We selected our decay function by running an extensive set of experiments with ResNet-50 on ImageNet (see Appendix C for details). We chose linear decay because it performed at least as well as all other schedules we tried, while also being the simplest and requiring only two additional metaparameters. In experiments that used linear decay, we specified metaparameters (η_0, α, T) such that the learning rate decayed linearly from η_0 to $\eta_T = \alpha\eta_0$. That is, the learning rate at step t is given by

$$\eta_t = \begin{cases} \eta_0 - (1-\alpha)\eta_0\frac{t}{T} & \text{if } t \leq T, \\ \alpha\eta_0 & \text{if } t > T. \end{cases}$$

Steps to result depends on the training metaparameters, and, for a given task and model, each batch size might have a different metaparameter configuration that minimizes steps to result. In all experiments, we independently tuned the metaparameters at each batch size, including the initial learning rate η_0 and, when learning rate decay was used, the decay schedule (α, T). Also, unless otherwise specified, we used the Nesterov momentum optimizer (Sutskever et al., 2013) and tuned the momentum γ.[12] Tuning anew for each batch size is extremely important since otherwise we would not be measuring steps to result as a function of batch size, rather we would be measuring steps to result as a function of batch size and the specific values of the learning rate and other metaparameters. We used quasi-random search (Bousquet et al., 2017) to tune the metaparameters with equal budgets of non-divergent[13] trials for different batch sizes. We selected metaparameter search spaces by hand based on preliminary experiments. The exact number of non-divergent trials needed to produce stable results depends on the search space, but 100 trials seemed to suffice in our experiments.[14] If the optimal trial occurred near the boundary of the search space, or if the goal validation error was not achieved within the search space, we repeated the search with a new search space. We measured steps to result for each batch size by selecting the metaparameter trial that reached the goal validation error in the fewest number of steps.

4.1. Steps to Result Depends on Batch Size in a Similar Way Across Problems

To get a sense of the basic empirical relationship, we measured the number of steps required to reach a goal validation error as a function of batch size across several different data sets and models (Figure 1). In all cases, as the batch size grows, there is an initial period of **perfect scaling** (b-fold benefit, indicated with a dashed line on the plots) where the steps needed to achieve the error goal halves for each doubling of the batch size. However, for all problems, this is followed by a region of **diminishing returns** that eventually leads to a regime of **maximal data parallelism** where additional parallelism provides no benefit whatsoever. In other words, for any given problem and without making strong assumptions about learning rates or other optimizer parameters, we can achieve both extremes suggested by theory (see Section 3.1.1). *A priori*, it is not obvious that every workload in our experiments should exhibit perfect scaling at the smallest batch sizes instead of immediately showing diminishing returns.

---

12. For LSTM for LM1B, we used a fixed value of $\gamma = 0.99$. We chose this value based on initial experiments and validated that tuning γ did not significantly affect the results for batch sizes 256, 1,024, or 4,096.
13. We discarded trials with a divergent training loss, which occurred when the learning rate was too high.
14. We used 100 non-divergent trials for all experiments except Transformer Shallow on LM1B with SGD, Transformer on Common Crawl, and LSTM on LM1B, for which we used 50 trials each.

(a) Simple CNN on MNIST   (b) Simple CNN on Fashion MNIST   (c) ResNet-8 on CIFAR-10

(d) ResNet-50 on ImageNet   (e) ResNet-50 on Open Images   (f) Transformer on LM1B

(g) Transformer on Common Crawl   (h) VGG-11 on ImageNet   (i) LSTM on LM1B
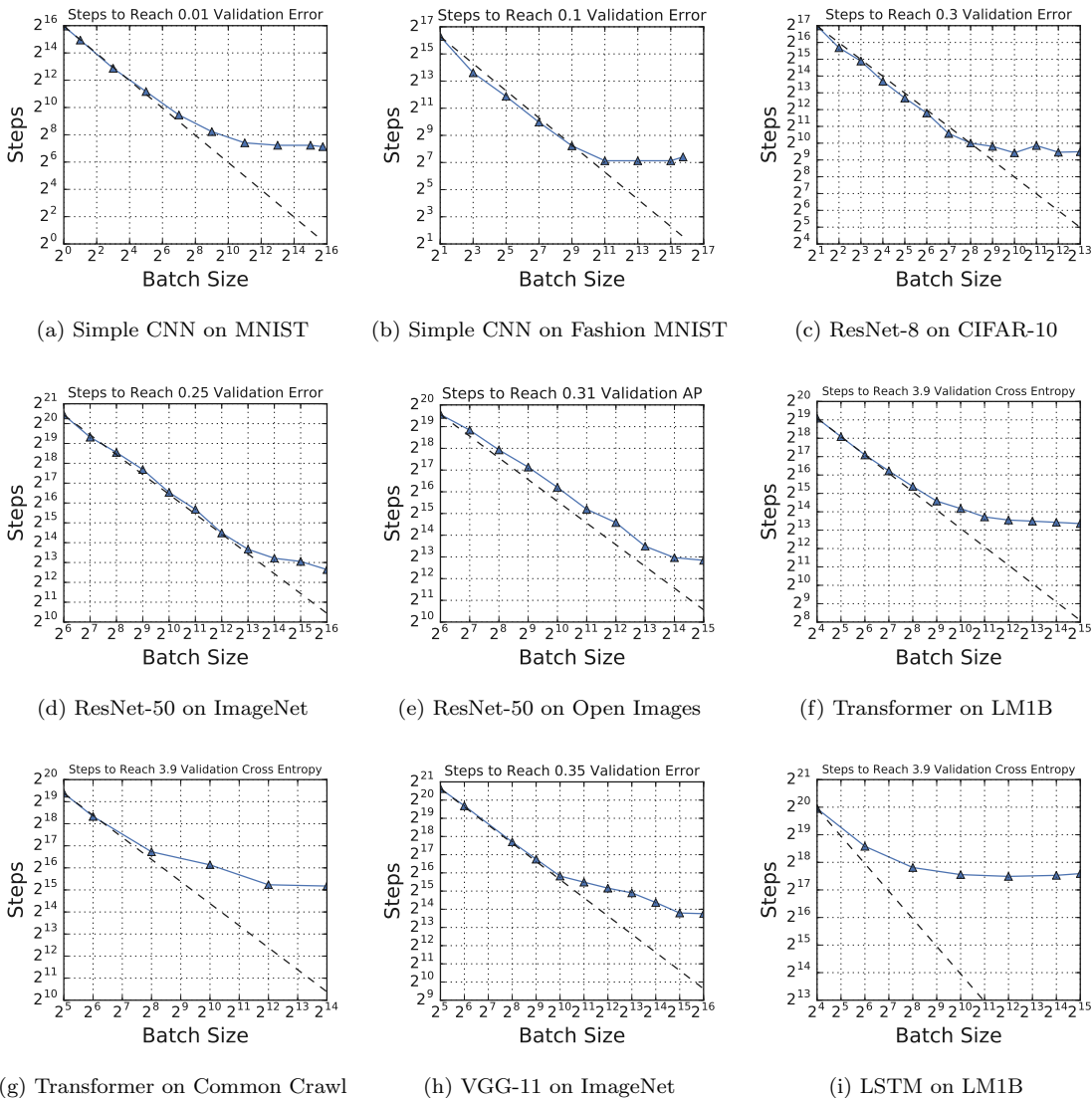
Figure 1: **The relationship between steps to result and batch size has the same characteristic form for all problems.** In all cases, as the batch size grows, there is an initial period of **perfect scaling** (indicated with a dashed line) where the steps needed to achieve the error goal halves for each doubling of the batch size. Then there is a region of **diminishing returns** that eventually leads to a region of **maximal data parallelism** where additional parallelism provides no benefit whatsoever. AP denotes average precision (see Appendix A).

(a) Simple CNN on MNIST   (b) Transformer on LM1B   (c) ResNet-50 on ImageNet
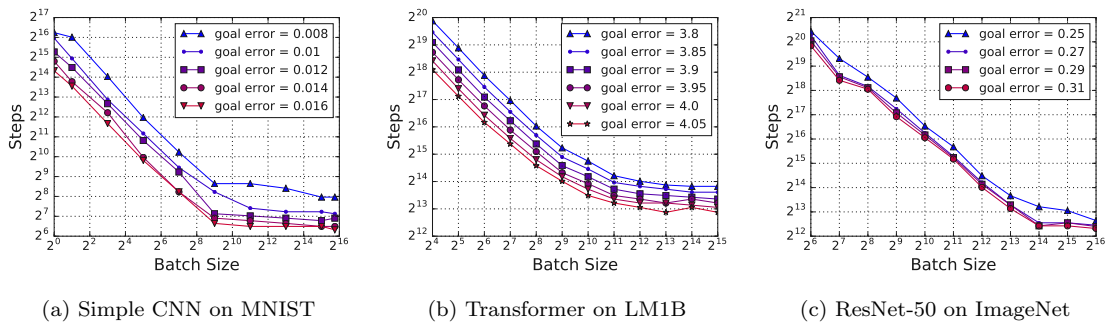
Figure 2: **Steps-to-result plots have a similar form for different (nearby) performance goals.** The transition points between the three regions (perfect scaling, diminishing returns, and maximal data parallelism) are nearly the same.

4.2. Validating Our Measurement Protocol

If the curves in Figure 1 were very sensitive to the goal validation error, then measuring the steps needed to reach our particular choice of the goal would not be a meaningful proxy for training speed. For small changes in the goal validation error, we do not care about vertical shifts as long as the transition points between the three scaling regions remain relatively unchanged. Figure 2 shows that varying the error goal only vertically shifts the steps-to-result curve, at least for modest variations centered around a good absolute validation error. Furthermore, although we ultimately care about out-of-sample error, if our plots looked very different when measuring the steps needed to reach a particular *training* error, then we would need to include both curves when presenting our results. However, switching to training error does not change the plots much at all (see Figure 12 in the Appendix).

Our experiments depend on extensive metaparameter tuning for the learning rate, momentum, and, where applicable, the learning rate schedule. For each experiment, we verified our metaparameter search space by checking that the optimal trial was not too close to a boundary of the space. See Figures 13 and 14 in the Appendix for examples of how we verified our search spaces.

4.3. Some Models Can Exploit Much Larger Batch Sizes Than Others

We investigated whether some models can make more use of larger batches than others by experimenting with different models while keeping the data set and optimizer fixed. We explored this question in two ways: (i) by testing completely different model architectures on the same data set, and (ii) by varying the size (width and depth) of a model within a particular model family. Since the absolute number of steps needed to reach a goal validation error depends on the model, the steps to result vs. batch size curves for each model generally appear at different vertical offsets from each other. Since we primarily care about the locations of the perfect scaling, diminishing returns, and maximal data parallelism regions, we normalized the y-axis of each plot by dividing by the number of steps needed to reach the goal for a particular batch size and data set. This normalization corresponds to a vertical shift of each curve (on log-scale plots), and makes it easier to compare different models. Appendix D contains all plots in this section without the y-axis normalized.
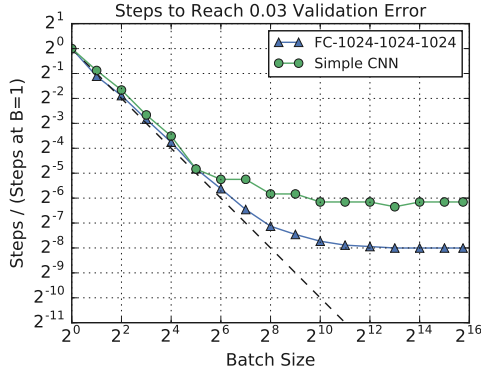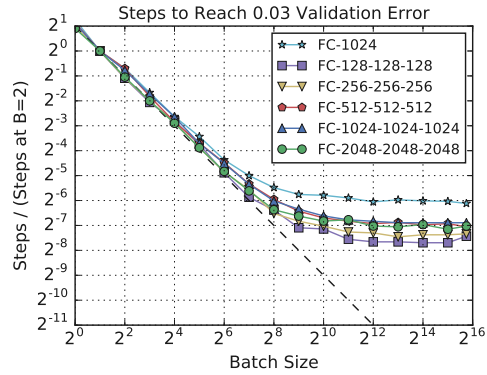
17

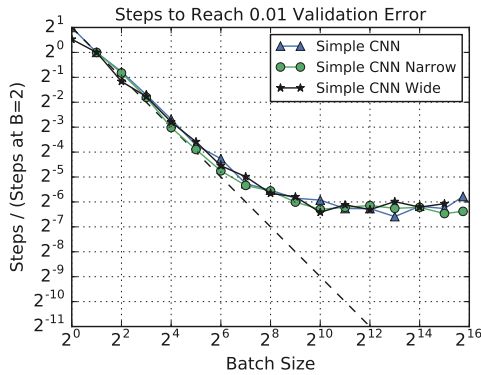(a) Fully Connected vs Simple CNN on MNIST
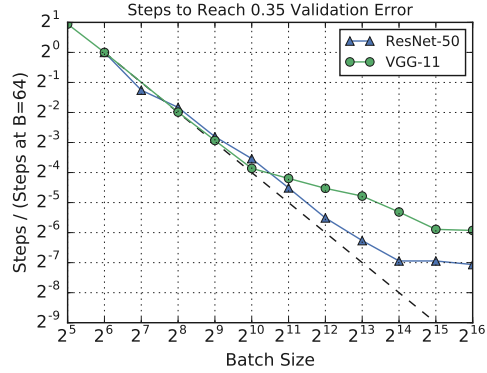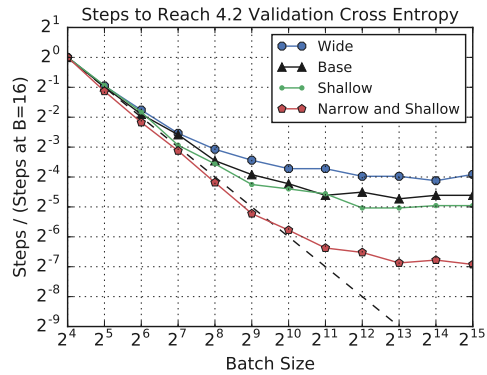
(b) ResNet-50 vs VGG-11 on ImageNet
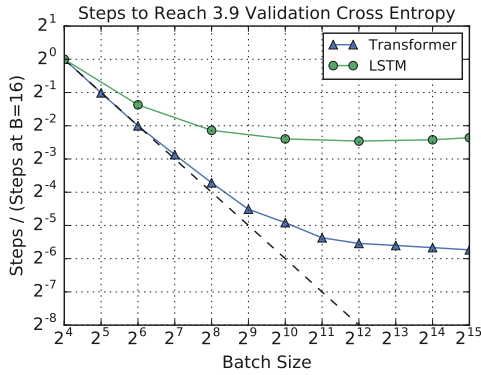
(c) Transformer vs LSTM on LM1B

(d) Fully Connected sizes on MNIST

(e) Simple CNN sizes on MNIST

(f) Transformer sizes on LM1B

Figure 3: **Some models can exploit much larger batch sizes than others.** Figures 3a-3c show that some model architectures can exploit much larger batch sizes than others on the same data set. Figures 3d-3f show that varying the depth and width can affect a model's ability to exploit larger batches, but not necessarily in a consistent way across different model architectures. All MNIST models in this Figure used plain mini-batch SGD, while all other models used Nesterov momentum. The goal validation error for each plot was chosen to allow all model variants to achieve that error. Figure 15 in the Appendix contains these plots without the y-axis normalized.

Figures 3a–3c show that the model architecture significantly affects the relationship between batch size and the number of steps needed to reach a goal validation error. In Figure 3a, the curve for the Fully Connected model flattens later than for the Simple CNN model on MNIST (although in this case the Simple CNN model can ultimately achieve better performance than the Fully Connected model). In Figure 3b, the curve for ResNet-50 flattens much later than the curve for VGG-11, indicating that ResNet-50 can make better use of large batch sizes on this data set. Unlike ResNet-50, VGG-11 does not use batch normalization or residual connections. Figure 3c shows that Transformer can make better use of large batch sizes than LSTM on LM1B.

Figures 3d–3f show that varying the depth and width can affect a model's ability to exploit larger batches, but not necessarily in a consistent way across different model architectures. In Figure 3d, the regions of perfect scaling, diminishing returns, and maximum useful batch size do not change much when the width is varied for the Fully Connected model on MNIST, although the shallower model seems less able to exploit larger batches than the deeper models. This contrasts with the findings of Chen et al. (2018), although they changed width and depth simultaneously while keeping the number of parameters fixed. For Simple CNN on MNIST, the relationship between batch size and steps to a goal validation error seems not to depend on width at all (Figure 15e in the Appendix shows that the curves are the same even when the y-axis is not normalized). However, in Figure 3f, the curves for *narrower* Transformer models on LM1B flatten later than for wider Transformer models, while the depth seems to have less of an effect. Thus, reducing width appears to allow Transformer to make more use of larger batch sizes on LM1B.

4.4. Momentum Extends Perfect Scaling to Larger Batch Sizes, but Matches Plain SGD at Small Batch Sizes

We investigated whether some optimizers can make better use of larger batches than others by experimenting with plain SGD, SGD with momentum, and Nesterov momentum on the same model and data set. Since plain SGD is a special case of both Nesterov momentum and SGD with momentum (with $\gamma = 0$ in each case), and since we tune γ in all experiments, we expect that experiments with either of these optimizers should do no worse than plain SGD at any batch size. However, it is not clear *a priori* whether momentum optimizers should outperform SGD, either by taking fewer training steps or by extending the perfect scaling region to larger batch sizes.

Figure 4 shows that Nesterov momentum and SGD with momentum can both extend the perfect scaling region beyond that achieved by SGD, and thus can significantly reduce the number of training steps required to reach a goal validation error at larger batch sizes. However, at batch sizes small enough that all optimizers are within their perfect scaling region, momentum optimizers perform identically to SGD without momentum. Though initially surprising, this identical performance at small batch sizes is consistent with observations made in Kidambi et al. (2018). In our experiments, we did not see a large difference between Nesterov momentum and SGD with momentum—Nesterov momentum appears to scale slightly better for Transformer on LM1B, but both perform about equally well for Simple CNN on MNIST.

(a) Simple CNN on MNIST   (b) Transformer Shallow on LM1B   (c) ResNet-8 on CIFAR-10
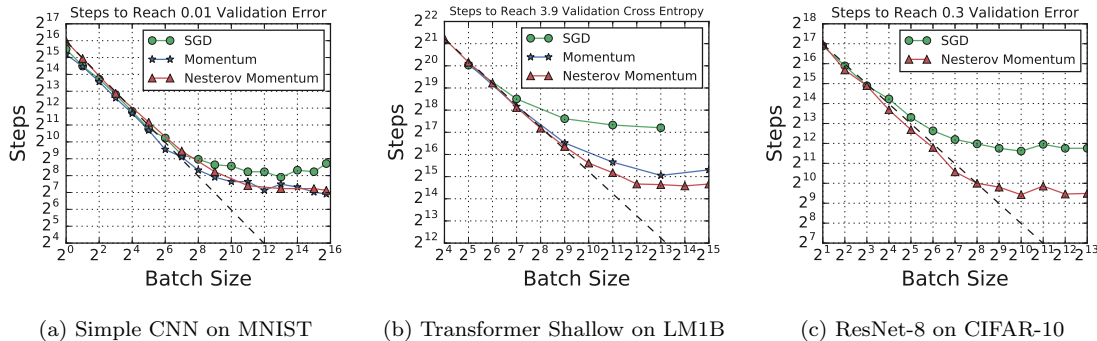
Figure 4: **Momentum extends perfect scaling to larger batch sizes, but matches plain SGD at small batch sizes.** Nesterov momentum and SGD with momentum can both significantly reduce the absolute number of training steps to reach a goal validation error, and also significantly extend the perfect scaling region and thus better exploit larger batches than plain mini-batch SGD.
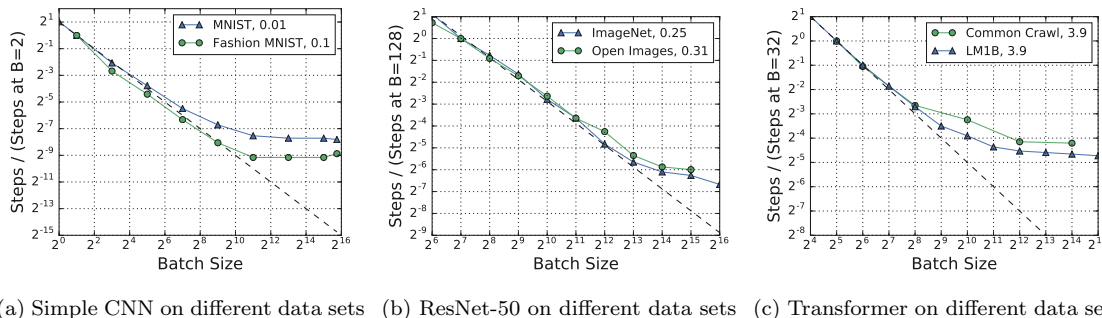

(a) Simple CNN on different data sets   (b) ResNet-50 on different data sets   (c) Transformer on different data sets

Figure 5: **The data set can influence the maximum useful batch size**. For the data sets shown in this plot, these differences are not simply as straightforward as larger data sets making larger batch sizes more valuable. Appendix A.2 describes the evaluation metric used for each data set, and the plot legends show the goal metric value for each task. Figure 16 in the Appendix contains these plots without the y-axis normalized.

4.5. The Data Set Matters, at Least Somewhat

We investigated whether properties of the data set make some problems able to exploit larger batch sizes than others by experimenting with different data sets while keeping the model and optimizer fixed. We approached this in two ways: (i) by testing the same model on completely different data sets, and (ii) by testing the same model on different subsets of the same data set. We normalized the y-axis of all plots in this section in the same way as Section 4.3. Appendix D contains all plots in this section without the y-axis normalized.

Figure 5 shows that changing the data set can affect the relationship between batch size and the number of steps needed to reach a goal validation error. Figure 5a shows that Fashion MNIST deviates from perfect scaling at a slightly larger batch size than MNIST for the Simple CNN model. Figure 5b shows that ImageNet and Open Images are extremely similar in how well ResNet-50 can make use of larger batch sizes, although, if anything, ImageNet might make slightly better use of larger batch sizes. Figure 5c shows that LM1B scales slightly better with increasing batch size than Common Crawl for Transformer. Since

(a) Simple CNN on MNIST subsets
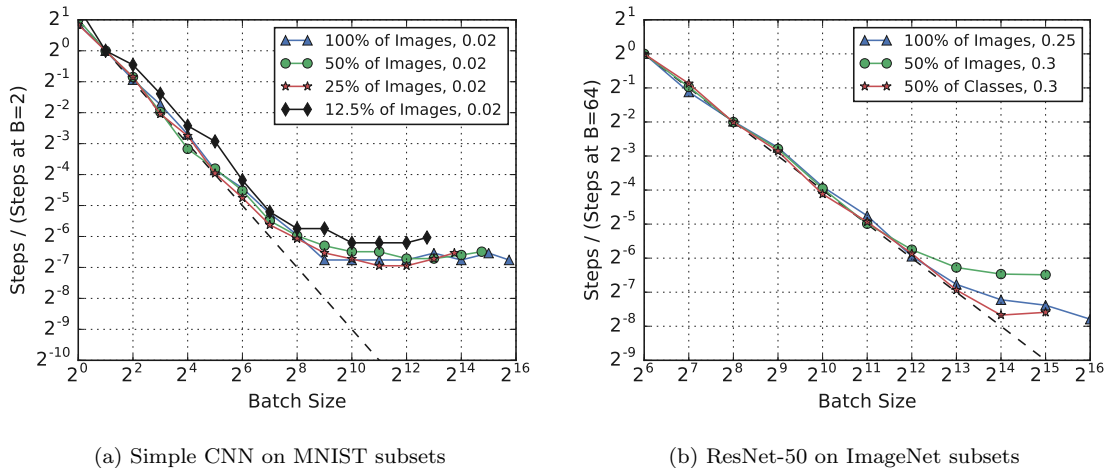
(b) ResNet-50 on ImageNet subsets

Figure 6: **Investigating the effect of data set size.** At least for MNIST, any effect of subset size on the maximum useful batch size is extremely small or nonexistent. For ImageNet, the random subset of half the images deviates from perfect scaling sooner than the full data set, but the curve for the subset with half the classes is very close to the curve for the full data set and, if anything, deviates from perfect scaling later. Appendix A.2 describes the evaluation metric used for each data set, and the plot legends show the goal metric value for each task. Figure 17 in the Appendix contains these plots without the y-axis normalized.

Fashion MNIST is the same size as MNIST, Open Images is larger than ImageNet, and Common Crawl is far larger than LM1B, these differences are not simply as straightforward as larger data sets making larger batch sizes more valuable.

To disentangle the effects from changes to the distribution and changes to the number of examples, we generated steps to result vs batch size plots for different random subsets of MNIST (Figure 6a) and ImageNet (Figure 6b). For MNIST, we selected subsets of different sizes, while for ImageNet, we selected a random subset of half the images and a similar sized subset that only includes images from half of the classes. At least on MNIST, any effect on the maximum useful batch size is extremely small or nonexistent. For ImageNet, Figure 6b shows that the random subset of half the images deviates from perfect scaling sooner than the full data set, but the curve for the subset with half the classes is very close to the curve for the full data set and, if anything, deviates from perfect scaling later, even though it contains roughly the same number of images as the random subset.

4.6. Regularization Can Be More Helpful at Some Batch Sizes Than Others

We used label smoothing (Szegedy et al., 2016) to regularize training in our experiments with ResNet-50 on ImageNet. Without label smoothing, we could not achieve our goal validation error rate of 0.25 with batch sizes greater than 2^{14} within our training budget. With a fixed compute budget for each batch size, label smoothing improved the error by as much as one percentage point at large batch sizes, while having no apparent effect at small batch sizes (Figure 7a). Meanwhile, if multiple choices for the label smoothing metaparameter achieved the goal within the training budget, then label smoothing did not change the number of steps needed (Figure 7b).
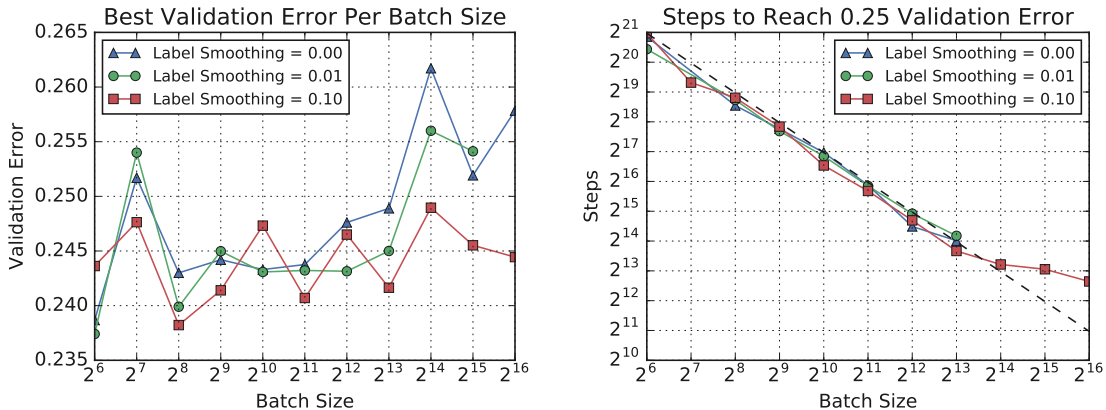
(a) Label smoothing benefits larger batch sizes, but has no apparent effect for smaller batch sizes.

(b) Label smoothing has no apparent effect on training speed, provided the goal error is achieved.

Figure 7: **Regularization can be more helpful at some batch sizes than others.** Plots are for ResNet-50 on ImageNet. Each point corresponds to a different metaparameter tuning trial, so the learning rate, Nesterov momentum, and learning rate schedule are independently chosen for each point. The training budget is fixed for each batch size, but varies between batch sizes.

We confirmed that label smoothing reduced overfitting at large batch sizes for ResNet-50 on ImageNet (see Figure 18 in the Appendix). This is consistent with the idea that noise from small batch training is a form of implicit regularization (e.g. Goodfellow et al., 2016). However, although our results show that *other forms of regularization can serve in place of this noise*, it might be difficult to select and tune other forms of regularization for large batch sizes. For example, we unsuccessfully tried to control overfitting with larger batch sizes by increasing the L_2 weight penalty and by applying additive Gaussian gradient noise before we obtained good results with label smoothing.

Finally, we also tried label smoothing with Simple CNN on MNIST and Fashion MNIST, and found that it generally helped *all* batch sizes, with no consistent trend of helping smaller or larger batch sizes more (see Figure 19 in the Appendix), perhaps because these data sets are sufficiently small and simple that overfitting is an issue at all batch sizes.

4.7. The Best Learning Rate and Momentum Vary with Batch Size

Across all problems we considered, the effective learning rate (η^{eff}; see Section 2.2) that minimized the number of training steps to a goal validation error tended to increase with increasing batch size (Figure 8). However, it did not always follow either a linear or square root scaling heuristic, despite the popularity of these rules of thumb. In some cases, the optimal effective learning rate even decreased for larger batch sizes. We also found that the best effective learning rate should be chosen by jointly tuning the learning rate and momentum, rather than tuning only the learning rate. For example, the optimal way to scale the effective learning rate for Transformer was to increase the momentum while decreasing the learning rate or holding it constant (see Figures 21 and 22 in the Appendix). This is a refinement to past prescriptions that only change the learning rate while keeping the momentum fixed.

(a) Simple CNN on MNIST   (b) Simple CNN on Fashion MNIST   (c) ResNet-8 on CIFAR-10

(d) ResNet-50 on ImageNet   (e) ResNet-50 on Open Images   (f) Transformer on LM1B

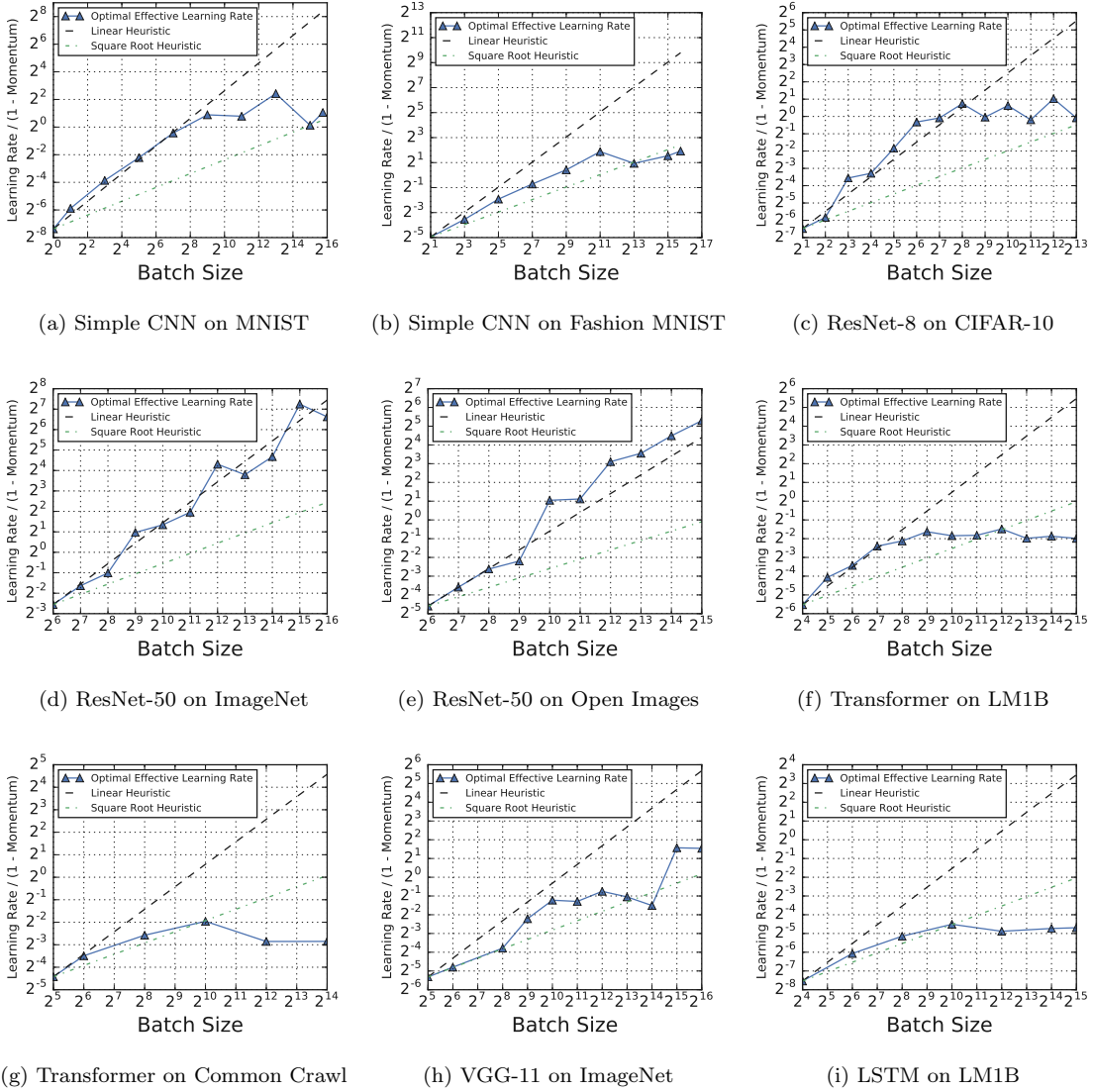(g) Transformer on Common Crawl   (h) VGG-11 on ImageNet   (i) LSTM on LM1B

Figure 8: **Optimal effective learning rates do not always follow linear or square root scaling heuristics.** Effective learning rates correspond to the trial that reached the goal validation error in the fewest training steps (see Figure 1). For models that used learning rate decay schedules (ResNet-8, ResNet-50, VGG-11), plots are based on the initial learning rate. See Figures 21 and 22 in the Appendix for separate plots of the optimal learning rate and momentum.
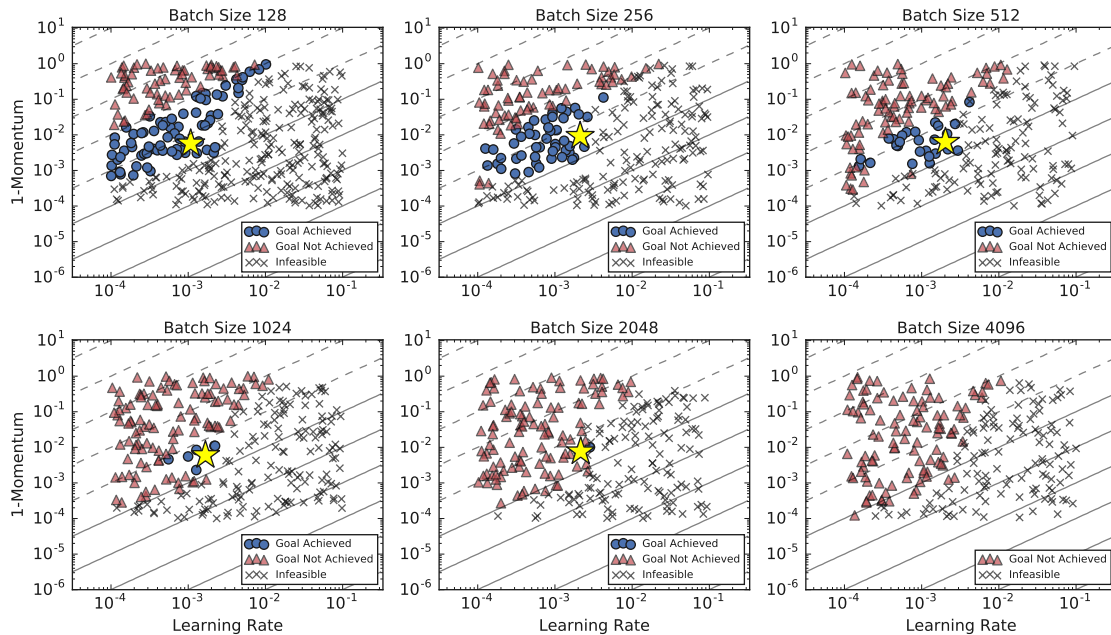
(a) Transformer on LM1B with a training budget of one epoch.
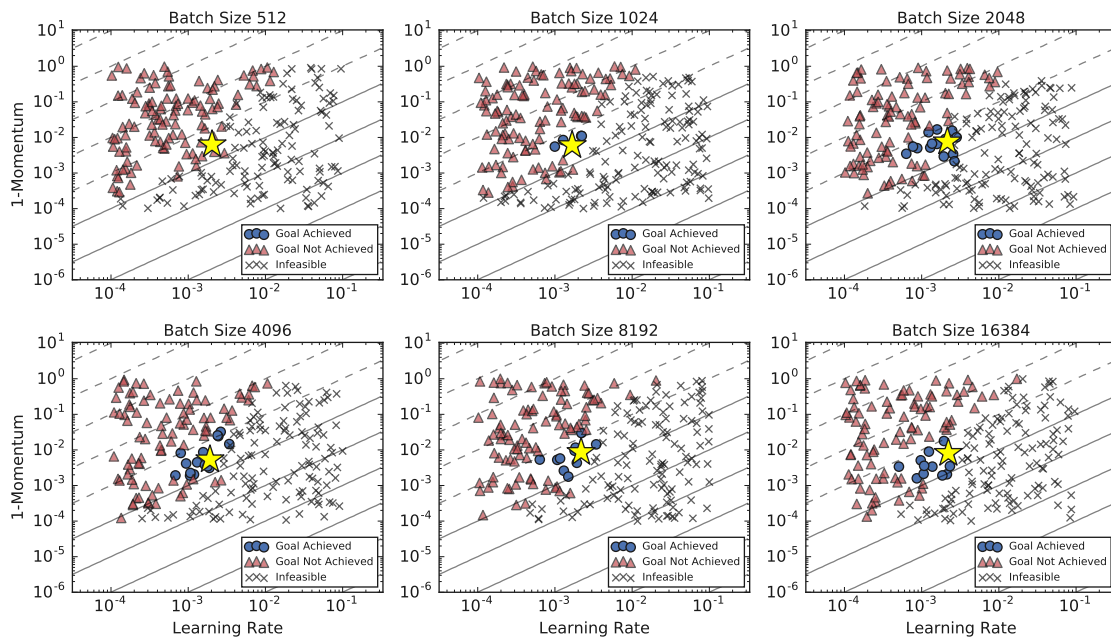
(b) Transformer on LM1B with a training budget of 25,000 steps.

Figure 9: **With increasing batch size, the region in metaparameter space corresponding to rapid training in terms of epochs becomes smaller, while the region in metaparameter space corresponding to rapid training in terms of step-count grows larger.** Yellow stars are the trials that achieved the goal in the fewest number of steps. Contours indicate the effective learning rate $\eta^{\mathrm{eff}} = \frac{\eta}{1-\gamma}$. Infeasible trials are those that resulted in divergent training.
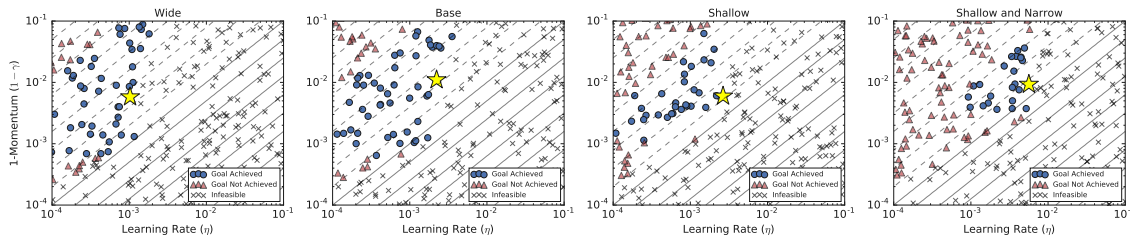
Figure 10: **Smaller models have larger stable learning rates for Transformer on LM1B.**
Plots are for different sizes of Transformer on LM1B with a batch size of 1024, a goal validation
cross entropy error of 4.2, and a training budget of 50,000 steps. Contours indicate the effective
learning rate $\eta^{\text{eff}} = \frac{\eta}{1-\gamma}$. Infeasible trials are those that resulted in divergent training.

We further investigated the relationship between learning rate, momentum, and training
speed by examining our metaparameter search spaces for different batch sizes and model
sizes. For this analysis, we used Transformer on LM1B with Nesterov momentum because
the metaparameter search spaces are consistent between all batch and model sizes, and
can be easily visualized because they consist only of the constant learning rate η and the
momentum γ. We observe the following behaviors:

- With increasing batch size, the region in metaparameter space corresponding to rapid
  training in terms of epochs becomes smaller (Figure 9a, consistent with the findings
  of Breuel, 2015b), while the region in metaparameter space corresponding to rapid
  training in terms of step-count grows larger (Figure 9b, although it eventually plateaus
  for batch sizes in the maximal data parallelism regime). Thus, with a fixed error goal
  and in a setting where training epochs are constrained (e.g. a compute budget), it may
  become more challenging to choose good values for the metaparameters with increasing
  batch size. Conversely, with a fixed error goal and in a setting where training steps
  are constrained (e.g. a wall-time budget), it may become easier to choose good values
  for the metaparameters with increasing batch size.

- The metaparameters yielding the fastest training are typically on the edge of the feasi-
  ble region of the search space (Figure 9). In other words, small changes in the optimal
  metaparameters might make training diverge. This behavior may pose a challenge for
  metaparameter optimization techniques, such as Gaussian Process approaches, that
  assume a smooth relationship between metaparameter values and model performance.
  It could motivate techniques such as learning rate warm-up that enable stability at
  larger eventual learning rates, since the maximum stable learning rate depends on the
  current model parameters. We did not observe the same behavior for ResNet-50 on
  ImageNet. Figure 20 in the Appendix shows the results for a range of effective learning
  rates near the optimum for ResNet-50 on ImageNet and Transformer on LM1B.

- Smaller models have larger stable learning rates (Figure 10). This is consistent with
  recent work predicting that the largest stable learning rate is inversely proportional
  to layer width (Karakida et al., 2018).

4.8. Solution Quality Depends on Compute Budget More Than Batch Size

We investigated the relationship between batch size and out-of-sample error for Simple CNN on MNIST and Fashion MNIST, and for two sizes of Transformer on LM1B. For each task, we ran a quasi-random metaparameter search over the constant learning rate η and Nesterov momentum γ. For MNIST and Fashion MNIST, we also added label smoothing and searched over the label smoothing parameter in $\{0, 0.1\}$ to mitigate any confounding effects of overfitting (see Section 4.6). We ran 100 metaparameter trials for each batch size with a large practical wall-time budget.

To disentangle the effects of the batch size from the compute budget, we compared batch sizes subject to budgets of either training steps or training epochs. For each batch size and compute budget, we found the model checkpoint that achieved the best validation accuracy across all metaparameter trials, and across all training steps that fell within the compute budget. Figure 11 shows the validation error for these best-validation-error checkpoints, as a function of batch size, for a range of compute budgets. We observe that, subject to a budget on training steps, larger batch sizes achieve better out-of-sample error than smaller batch sizes, but subject to a budget on training epochs, smaller batch sizes achieve better out-of-sample error than larger batch sizes. These observations are likely explained by the observations that, for a fixed number of training steps, larger batch sizes train on more data, while for a fixed number of epochs, smaller batch sizes perform more training steps.

The workloads in Figure 11 represent two distinct modes of neural network training. For the small MNIST and Fashion MNIST data sets, we used training budgets that would saturate (or almost saturate) performance at each batch size. In other words, out-of-sample error cannot be improved by simply increasing the budget, with caveats due to practical limitations on our ability to find optimal values for the metaparameters. Figures 11a and 11b show that differences in maximum performance between batch sizes on these data sets are very small (see Figures 23 and 24 in the Appendix for zoomed versions of these plots). We cannot rule out that any differences at this magnitude are due to noise from metaparameter choices and training stochasticity. Thus, for these workloads at least, the effect of batch size on solution quality is either very small or nonexistent. On the other hand, we cannot saturate performance with Transformer on LM1B within a practical training time. In this case, Figures 11c and 11d show that the best error is simply achieved by the largest compute budget.

Taken together, these observations suggest that in practice *the relevant question is not which batch size leads to the best performance, but rather how compute budget varies as a function of batch size.* Although we tried our best to saturate performance with MNIST and Fashion MNIST, we found that it took millions of training steps for small batch sizes, and thousands of epochs for large batch sizes, even for data sets as small and simple as these. Indeed, despite sampling 100 metaparameter configurations per batch size and training for up to 25 hours per configuration, it is still not certain whether we truly saturated performance at the smallest and largest batch sizes (see Figures 23 and 24 in the Appendix). Thus, the regime of saturated performance is of limited practical concern for most workloads— the compute budget required to saturate performance is likely beyond what a practitioner would typically use. For realistic workloads, practitioners should be most concerned with identifying the batch size at which they can most efficiently apply their compute.

<table>
<tr><td>Step budget</td><td>Epoch budget</td></tr>
</table>

(a) Simple CNN on MNIST

(b) Simple CNN on Fashion MNIST

(c) Transformer (narrow and shallow) on LM1B
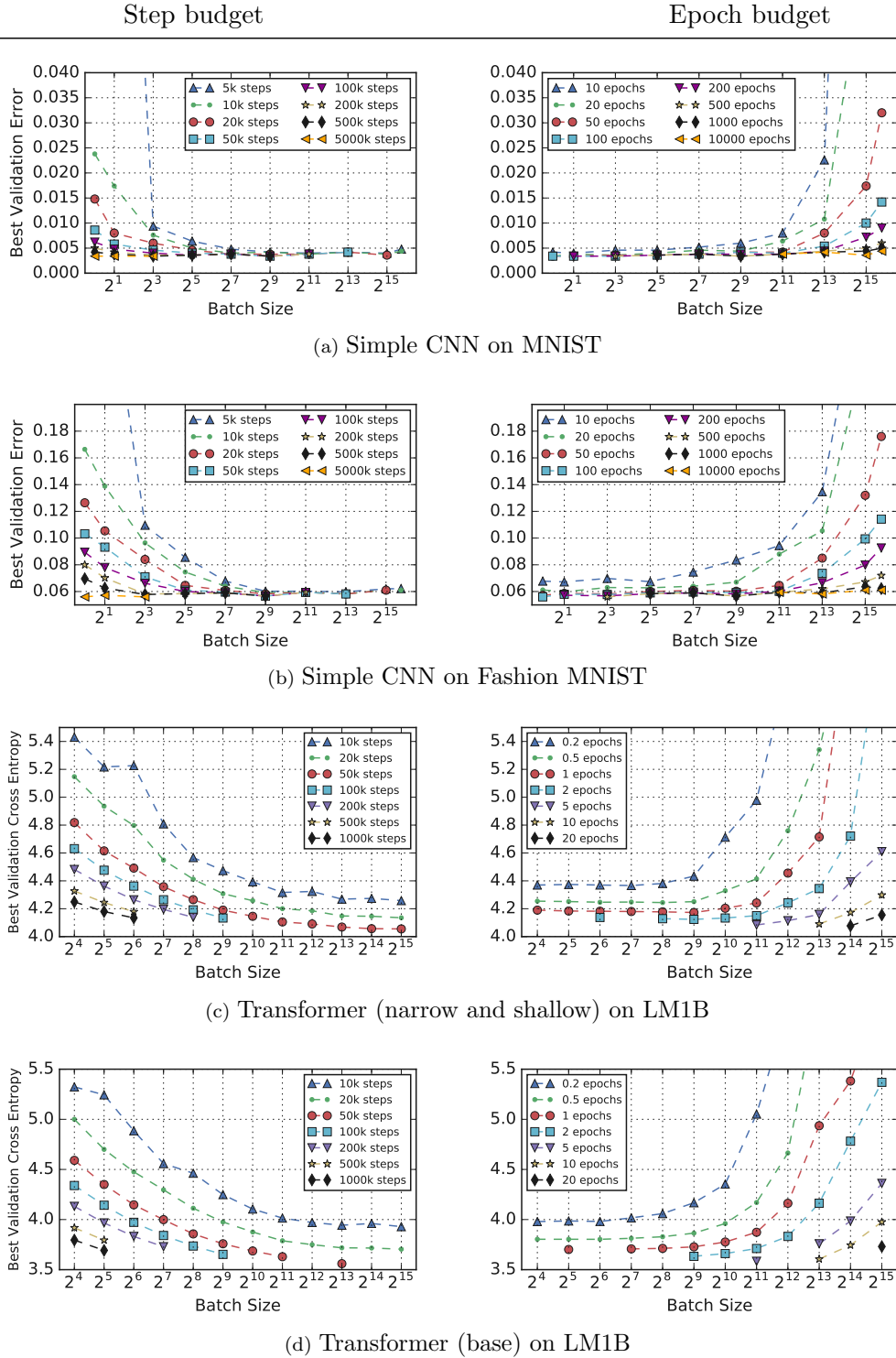

(d) Transformer (base) on LM1B

Figure 11: **Validation error depends on compute budget more than batch size.** Plots show the best validation error subject to budgets of training steps (left column) or training epochs (right column). Step budgets favor large batch sizes, while epoch budgets favor small batch sizes.

5. Discussion

Our goals in measuring the effects of data parallelism on neural network training were twofold: first, we hoped to produce actionable advice for practitioners, and second, we hoped to understand the utility of building systems capable of very high degrees of data parallelism. Our results indicate that, for idealized data parallel hardware, there is a universal relationship between training time and batch size, but there is dramatic variation in how well different workloads can make use of larger batch sizes. Across all our experiments, increasing the batch size initially reduced the number of training steps needed proportionally. However, depending on the workload, this perfect scaling regime ended anywhere from a batch size of 2^4 to a batch size of 2^{13}. As batch size increases beyond the perfect scaling regime, there are diminishing returns (where increasing the batch size by a factor of k only reduces the number of training steps needed by a factor less than k) that end with a maximum useful batch size (where increasing the batch size no longer changes the number of training steps needed). Once again, the maximum useful batch size is extremely problem-dependent and varied between roughly 2^9 and 2^{16} in our experiments. Other workloads may have the region of perfect scaling end at batch sizes even smaller or larger than the range we observed, as well as having even smaller or larger maximum useful batch sizes.

On the one hand, the possibility that perfect scaling can extend to batch sizes beyond 2^{13} for some workloads is good news for practitioners because it suggests that efficient data-parallel systems can provide extremely large speedups for neural network training. On the other hand, the wide variation in scaling behavior across workloads is bad news because any given workload might have a maximum useful batch size well below the limits of our hardware. Moreover, for a new workload, measuring the training steps needed as a function of batch size and confirming the boundaries of the three basic scaling regimes requires expensive experiments. In this work, we have only described how to retrospectively predict the scaling behavior by tuning the optimization metaparameters for every batch size. Although Golmant et al. (2018) also described the same basic scaling behavior we found, in their experiments the relationship did not appear consistently across problems, across error goals, or in out-of-sample error. In light of our own results, the heuristics they assumed for adjusting the learning rate as a function of batch size are the likely cause of these inconsistencies, but this explanation only drives home the inconvenience of having to carefully tune at every new batch size. We were unable to find reliable support for any of the previously proposed heuristics for adjusting the learning rate as a function of batch size. Thus we are forced to recommend that practitioners tune all optimization parameters anew when they change the batch size or they risk masking the true behavior of the training procedure.

If the scaling behavior of workloads with respect to batch size has a simple dependence on properties of the workload, then we might be able to predict the limits of perfect scaling (or the maximum useful batch size) before running extensive experiments. We could then prioritize workloads to run on specialized hardware or decide whether gaining access to specialized hardware would be useful for a given workload of interest. On the one hand, our results are bad news for practitioners because they show that accurate scaling predictions must depend on a combination of non-obvious properties of the model, optimizer, and data set. On the other hand, we have a lot of control over the choice of model and optimizer

and there is some indication that they might be responsible for the largest portion of the variation between workloads. Our results comparing SGD and SGD with momentum (or Nesterov momentum) show that, at least for the problems we tried, momentum can extend perfect scaling to much larger batch sizes, offering clear guidance for practitioners. Other optimizers, such as KFAC (Martens and Grosse, 2015; Grosse and Martens, 2016; Ba et al., 2017), or optimization techniques designed specifically for massively data parallel systems (e.g. Li et al., 2014), might allow perfect scaling to extend much further. Intuitively, it seems plausible that optimizers that estimate local curvature information might be able to benefit more from large batches than optimizers that only use gradients.

Although the model seems to have a large effect on the maximum useful batch size and the limit of perfect scaling, our results do not give definitive answers on exactly how to design models that scale better for a given optimizer and data set. Even when we kept the model family fixed, we observed somewhat inconsistent results from changing the model width and depth. Chen et al. (2018) suggested that wider models can exploit larger batch sizes than narrower models, but their theoretical arguments only apply to linear networks and fully connected networks with a single hidden layer. In contrast, we found that *narrower* variants of the Transformer model scaled better to larger batch sizes, although it is unclear if the same notion of "width" transfers between different types of neural networks.

Unlike the model and optimizer, we generally have much less control over the data set. Unfortunately, properties of the data set also affect how well training scales in practice. Our results are equivocal on whether the number of training examples has any effect, but changing the data set entirely can certainly change the scaling behavior with respect to batch size.

Finally, our results at least partially reconcile conflicting stances in the literature on whether increasing the batch size degrades model quality. Our experiments show that:

1. Any study that only tunes the learning rate for one batch size and then uses a heuristic to choose the learning rate for other batch sizes (Goyal et al., 2017; Keskar et al., 2017; Hoffer et al., 2017; Lin et al., 2018; Devarakonda et al., 2017; Golmant et al., 2018) gives a systematic advantage to the batch size used in tuning (as well as nearby batch sizes). Our results did not show a simple relationship between the optimal learning rate and batch size that scales indefinitely (see Figures 8 and 21), so the use of simple heuristics for batch sizes sufficiently far from the base batch size could very well explain the degraded solutions and divergent training reported in prior work. Similarly, the optimal values of other metaparameters, such as the momentum and learning rate decay schedule, should not be assumed to remain constant or scale in a simple way as the batch size increases.

2. Assuming an epoch budget when comparing solution quality between batch sizes (Masters and Luschi, 2018; Goyal et al., 2017; Lin et al., 2018; Devarakonda et al., 2017), in effect, limits an investigation to the perfect scaling region of the steps to result vs batch size curve (see Figure 1). This budget favors smaller batch sizes because they will perform more optimizer steps for the same number of training examples (see Section 4.8). Certainly, there are situations where an epoch budget is appropriate, but there may exist budgets just outside the perfect scaling region that can achieve the same quality solution, and those budgets may still represent a significant reduction

in the number of training steps required. Moreover, even for a fixed model and data set, simply changing the optimizer can significantly extend the perfect scaling regime to larger batch sizes. For example, Masters and Luschi (2018) found that test performance of ResNet-8 (without batch normalization) on CIFAR-10 with a fixed epoch budget degraded after batch size 16, but considered only plain mini-batch SGD. Our experiments confirmed that perfect scaling ends at batch size 16 with plain mini-batch SGD, but using Nesterov momentum extends the perfect scaling regime to batch size 256 (see Figure 1c).

3. Assuming a step budget when comparing solution quality between batch sizes (Hoffer et al., 2017) might favor larger batch sizes because they will see more training examples for the same number of gradient updates (see Section 4.8). A step budget is likely sufficient for a larger batch size to reach *at least* the same performance as a smaller batch size: we never saw the number of steps to reach a goal validation error increase when the batch size was increased (see Figure 1).

4. Increasing the batch size reduces noise in the gradient estimates (see Equation 4). However, the noise in updates due to small batches might, in some cases, provide a helpful regularization effect (Goodfellow et al., 2016; Smith and Le, 2018). Thankfully, other regularization techniques, such as label smoothing, can replace this effect (see Section 4.6). Others have also used regularization techniques, such as data augmentation (Keskar et al., 2017) and L_2 regularization (Smith and Le, 2018), to eliminate the "generalization gap" between two batch sizes.

5. Finally, although we do not believe there is an inherent degradation in solution quality associated with increasing the batch size, depending on the compute budget, it may become increasingly difficult to find good values for the metaparameters with larger batch sizes. Specifically, increasing the batch size may shrink the region in metaparameter space corresponding to rapid training in terms of epochs (see Figure 9a), as previously reported by Breuel (2015b). On the other hand, increasing the batch size may increase the region in metaparameter space corresponding to rapid training in terms of steps (see Figure 9b).

5.1. Limitations of our experimental protocol

When interpreting our results, one should keep in mind any limitations of our experimental protocol. We do not believe any of these limitations are debilitating, and we hope that describing these potential areas of concern will spur methodological innovation in future work.

Firstly, we were unable to avoid some amount of human judgment when tuning metaparameters. Although we did not tune metaparameters by hand, we specified the search spaces for automatic tuning by hand and they may not have been equally appropriate for all batch sizes, despite our best efforts. We are most confident in our search spaces that tuned the fewest metaparameters (such as in our experiments that only tuned learning rate and momentum). We found it quite difficult to be confident that our tuning was sufficient when we searched over learning rate decay schedules; readers should be aware that the steps to result measurement is generally quite sensitive to the learning rate schedule. Thus, we

may not have sampled enough trials at some batch sizes or, nearly equivalently, our search spaces may have been too wide at some batch sizes. Even though we verified that the best trial was not on the boundary of the search space, this by no means guarantees that we found the globally optimal metaparameters.

Smaller batch sizes typically had more opportunities to measure validation error and, when validation error was noisy, got more chances to sample a lucky validation error. Batch sizes (usually larger ones) that did not reach the goal validation error using the first search space used revised search spaces that gave them an extra bite of the apple, so to speak.

Finally, our analysis does not consider how robustly we can reach a goal error rate. For instance, we did not distinguish between batch sizes where all 100 trials achieved the goal validation error and batch sizes where only one of the 100 trials achieved the goal. The maximum or minimum value over a set of trials is not usually a very robust statistic, but something like the 50th percentile trial mostly reveals information about the search space. We tried to strike a balance between studying realistic workloads and being able to repeat our experiments so many times that these uncertainty questions became trivial. Ultimately, we opted to study realistic workloads and simply report results for the optimal trials.

6. Conclusions and Future Work

Increasing the batch size is a simple way to produce valuable speedups across a range of workloads, but, for all workloads we tried, the benefits diminished well within the limits of current hardware. Unfortunately, blindly increasing the batch size to the hardware limit will not produce a large speedup for all workloads. However, our results suggest that some optimization algorithms may be able to consistently extend perfect scaling across many models and data sets. Future work should perform our same measurements with other optimizers, beyond the closely-related ones we tried, to see if any existing optimizer extends perfect scaling across many problems. Alternatively, if we only need speedups for specific, high-value problems, we could also consider designing models that extend perfect scaling to much larger batch sizes. However, unlike the optimizer, practitioners are likely to tailor their model architectures to the specific problems at hand. Therefore, instead of searching for model architectures that happen to scale extremely well, future work should try to uncover general principles for designing models that can scale perfectly to larger batch sizes. Even if such principles remain elusive, we would still benefit from methods to prospectively predict the scaling behavior of a given workload without requiring careful metaparameter tuning at several different batch sizes. Finally, the deep learning community can always benefit from methodical experiments designed to test hypotheses, characterize phenomena, and reduce confusion, to balance more exploratory work designed to generate new ideas for algorithms and models.

Appendix A. Data Set Details

This section contains details of the data sets summarized in Table 1.

A.1. Data Set Descriptions and Pre-Processing

MNIST (LeCun et al., 1998) is a classic handwritten digit image classification data set with 10 mutually exclusive classes. We split the original training set into 55,000 training images and 5,000 validation images, and used the official test set of 10,000 images. We did not use data augmentation.

Fashion MNIST (Xiao et al., 2017) is another reasonably simple image classification data set with 10 mutually exclusive classes. It was designed as a drop-in replacement for MNIST. We split the original training set into 55,000 training images and 5,000 validation images, and used the official test set of 10,000 images. We did not use data augmentation.

CIFAR-10 (Krizhevsky, 2009) is an image classification data set of 32×32 color images with 10 mutually exclusive classes. We split the original training set into 45,000 training images and 5,000 validation images. We used the official test set of 10,000 images. We pre-processed each image by subtracting the average value across all pixels and channels and dividing by the standard deviation.[15] We did not use data augmentation.

ImageNet (Russakovsky et al., 2015) is an image classification data set with 1,000 mutually exclusive classes. We split the official training set into 1,281,167 training images and 50,045 test images, and used the official validation set of 50,000 images. We pre-processed the images and performed data augmentation in a similar way to Simonyan and Zisserman (2014). Specifically, at training time, we sampled a random integer $S \in [256, 512]$, performed an aspect-preserving resize so that the smallest side had length S, and took a random crop of size $(224, 224)$. We randomly reflected the images horizontally, but unlike Simonyan and Zisserman (2014), we did not distort the colors. At evaluation time, we performed an aspect-preserving resize so that the smallest side had length 256, and took a central crop of size $(224, 224)$. In both training and evaluation, we then subtracted the global mean RGB value from each pixel using the values computed by Simonyan and Zisserman (2014).[16]

Open Images v4 (Krasin et al., 2017) is a data set of 9 million images that are annotated with image-level labels and object bounding boxes.[17] The image labels were generated by a computer vision model and then verified as either *positive* or *negative* labels by human annotators. We only considered the 7,186 "trainable" classes with at least 100 human-annotated positives in the training set. We filtered the official subsets by keeping only images with at least one positive trainable label, which produced training, validation and test sets of size 4,526,492; 41,225; and 124,293 images, respectively. On average, each image in the training set has 2.9 human-annotated positive labels, while each image in the validation and test sets have 8.4 human-annotated positive labels. We only considered the human-annotated positives and assumed all other classes were negative. We pre-processed the images and performed data augmentation identically to ImageNet.

---

15. We used the TensorFlow op `tf.image.per_image_standardization`.
16. See `https://gist.github.com/ksimonyan/211839e770f7b538e2d8#description` for the mean RGB values used.
17. Available at `https://storage.googleapis.com/openimages/web/index.html`.

LM1B (Chelba et al., 2014) is a text data set of English news articles.[18] We used the official training set and created validation and test sets using files `news.en.heldout-00000-of-00050` and `news.en.heldout-00001-of-00050`, respectively. These splits contain 30,301,028; 6,075; and 6,206 sentences, respectively. We used an invertable word tokenizer to split the text into sub-word tokens with a vocabulary of size 32,000.[19] On average, the training set contains around 20 tokens per sentence and the validation and test sets contain around 29 tokens per sentence. At training time, we clipped long sentences to the first 64 tokens, which affected only about 2% of sentences. We did not clip long sentences at evaluation time. The maximum sentence across the validation and test sets has 476 tokens.

Common Crawl is a repository of web data containing over 3 billion web pages.[20] We filtered and processed the data set identically to Anil et al. (2018).[21] The vocabulary contains 24,006 sub-word tokens. We randomly partitioned the sentences into a training set (99.98%) and a holdout set (0.02%). Our training set contains ~25.8 billion sentences. We used the first 6,075 sentences of the holdout set as our validation set, which is the same number of sentences in our LM1B validation set. Some sentences are tens of thousands of tokens long. To maintain consistency with our LM1B processing, we clipped sentences to 64 tokens at training time and 476 at evaluation time.

A.2. Evaluation Metrics

We use **classification error** for MNIST, Fashion MNIST, CIFAR-10, and ImageNet. To compute this metric, we consider the model's classification for each image to be the class it assigns the highest probability. Then

$$\text{classification error} = \frac{\text{\# incorrect classifications}}{\text{\# classifications}}.$$

We use class-agnostic **average precision** (AP) for Open Images. To compute this metric, we first rank each image-class pair by the predicted likelihood of the class being a true positive for that image. Then

$$AP = \frac{1}{w} \sum_{k=1}^{nm} \text{Precision}(k) \cdot \text{Relevance}(k), \tag{7}$$

where $\text{Precision}(k)$ is the precision when considering the top k image-class pairs, $\text{Relevance}(k)$ is an indicator function equal to 1 if the kth image-class pair is a verified positive and 0 otherwise, n is the number of images in the validation set, m is the number of classes, and w is the number of positive labels. Average precision was proposed for Open Images by Veit et al. (2017). Due to false negatives in the validation set, Veit et al. (2017) only computed AP over the the human-annotated classes in each image. However, on average, each image

---

18. Available at `http://www.statmt.org/lm-benchmark/`.
19. The code for processing the raw data and generating the vocabulary is available at `https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/data_generators/lm1b.py`
20. Available at `http://commoncrawl.org/2017/07/june-2017-crawl-archive-now-available/`.
21. See `https://github.com/google-research/google-research/tree/master/codistillation` for document IDs.

in the validation set only has 8.4 positive and 4 negative human-annotated classes, so each image is only evaluated over $\sim$12 classes out of 7,186. This yields misleadingly high values of AP. Instead, we compute AP over all classes in each image, which may underestimate the true AP due to false negatives in the validation set, but is more indicative of the true performance in our experience. We compute AP using an efficient approximation of the area under the discrete precision-recall curve.[22]

We use average per-token **cross entropy error** for LM1B and Common Crawl. For a single sentence $s = (w_1, ..., w_m)$, let $p(w_j|w_1, ..., w_{j-1})$ denote the model's predicted probability of the token w_j given all prior tokens in the sentence. Thus, the predicted log-probability of s is $\log p(s) = \sum_{j=1}^{m} \log p(w_j|w_1, ..., w_{j-1})$. We compute the average per-token cross entropy error over a data set $\{s_1, ..., s_n\}$ as

$$\text{cross entropy error} = \frac{\sum_{i=1}^{n} \log p(s_n)}{\sum_{i=1}^{n} \text{len}(s_n)},$$

where $\text{len}(s)$ denotes the number of tokens in s. This is the logarithm of the per-token perplexity.

Appendix B. Model Details

In this section we give the architectural details of the models summarized in Table 2. In addition to the descriptions below, each model has a task-specific output layer. Models trained on MNIST, Fashion MNIST, CIFAR-10, and ImageNet (classification with mutually exclusive labels) use a softmax output layer to model the probability distribution over classes. Models trained on Open Images (classification with multiple labels per image) use a sigmoid output layer to model the probability of each class. Models trained on LM1B and Common Crawl (language modeling) use a softmax output layer to model the probability of the next word in a sentence given all prior words in the sentence.

Fully Connected is a fully connected neural network with ReLU activation function. Hidden layers use dropout with probability 0.4 during training. We vary the number of layers and number of units per layer in different experiments to investigate the impact of model size. We use the notation FC-N_1-...-N_k to denote a fully connected neural network with k hidden layers and N_i units in the ith layer.

Simple CNN consists of 2 convolutional layers with max-pooling followed by 1 fully connected hidden layer. The convolutional layers use 5×5 filters with stride length 1, "same" padding (Goodfellow et al., 2016), and ReLU activation function. Max pooling uses 2×2 windows with stride length 2. The fully connected layer uses dropout with probability 0.4 during training. We used three different model sizes: **base** has 32 and 64 filters in the convolutional layers and 1,024 units in the fully connected layer; **narrow** has 16 and 32 filters in the convolutional layers and 512 units in the fully connected layer; and **wide** has 64 and 128 filters in the convolutional layers and 2,048 units in the fully connected layer. We used the base model unless otherwise specified.

---

22. Equation 7 can be interpreted as a right Riemann sum of the discrete precision-recall curve $\{(r_i, p_i)|i = 1, ..., w\}$, where $r_i = i/w$ and p_i is the maximum precision among all values of precision with recall r_i (each value of recall may correspond to different values of precision at different classification thresholds). We use the TensorFlow op `tf.metrics.auc` with `curve="PR"`, `num_thresholds=200`, and `summation_method="careful_interpolation"`.

ResNet-8 consists of 7 convolutional layers with residual connections followed by 1 fully connected hidden layer. We used the model described in section 4.2 of He et al. (2016a) with $n = 1$, but with the improved residual block described by He et al. (2016b). We removed batch normalization, which is consistent with Masters and Luschi (2018).

ResNet-50 consists of 49 convolutional layers with residual connections followed by 1 fully connected hidden layer. We used the model described in section 4.1 of He et al. (2016a), but with the improved residual block described by (He et al., 2016b). We replaced batch normalization (Ioffe and Szegedy, 2015) with ghost batch normalization to keep the training objective fixed between batch sizes and to avoid possible negative effects from computing batch normalization statistics over a large number of examples (Hoffer et al., 2017). We used a ghost batch size of 32 for all experiments. We also applied label smoothing (Szegedy et al., 2016) to regularize the model at training time, which was helpful for larger batch sizes. The label smoothing coefficient was a metaparameter that we tuned in our experiments.

VGG-11 consists of 8 convolutional layers followed by 3 fully connected hidden layers. We used the model referred to as "model A" by Simonyan and Zisserman (2014).

LSTM is a one hidden-layer LSTM model (Hochreiter and Schmidhuber, 1997). It is a simpler variant of the LSTM-2048-512 model described by Jozefowicz et al. (2016), with 1,024 embedding dimensions, 2,048 hidden units, and 512 projection dimensions. We did not use bias parameters in the output layer because we found this improved performance in our preliminary experiments.

Transformer is a self-attention model that was originally presented for machine translation (Vaswani et al., 2017). We used it as an autoregressive language model by applying the decoder directly to the sequence of word embeddings for each sentence. We used four different sizes: the **base** model described by Vaswani et al. (2017); a **shallow** model that is identical to the base model except with only two hidden layers instead of six; a **narrow and shallow** model that is identical to the shallow model except with half as many hidden units and attention heads as well as half the filter size; and a **wide** model that is identical to the base model except with double the number of hidden units and attention heads as well as double the filter size. We used the base model unless otherwise specified.

Appendix C. Learning Rate Schedules

We chose our learning rate schedule by experimenting with a variety of different schedules for ResNet-50 on ImageNet. For each schedule, we specified the following metaparameters:

- η_0: initial learning rate

- α: decay factor $(\alpha > 0)$

- T: number of training steps until the learning rate decays from η_0 to $\alpha\eta_0$

Each schedule corresponds to a decay function $d(t)$, such that the learning rate at training step t is

$$\eta(t) = \begin{cases} d(t) \cdot \eta_0 & \text{if } t \leq T, \\ \alpha\eta_0 & \text{if } t > T. \end{cases}$$

We experimented with the following decay functions:

- **Constant**: $d(t) = 1$

- **Linear**: $d(t) = 1 - (1 - \alpha)\frac{t}{T}$

- **Cosine** (Loshchilov and Hutter, 2017): $d(t) = \alpha + \frac{(1-\alpha)}{2}\left(1 + \cos\pi\frac{t}{T}\right)$

- **Exponential Polynomial**: $d(t) = \alpha + (1 - \alpha)\left(1 - \frac{t}{T}\right)^{\lambda}$, where $\lambda > 0$

- **Inverse Exponential Polynomial**: $d(t) = \frac{\alpha}{\alpha + (1-\alpha)\left(\frac{t}{T}\right)^{\lambda}}$, where $\lambda > 0$

- **Exponential**: $d(t) = \alpha^{t/T}$

We also tried piecewise linear learning rate schedules. These schedules are specified by a sequence of pairs $\{(t_0, \eta_0), ..., (t_k, \eta_k)\}$, with $0 = t_0 < t_1... < t_k$, such that the learning rate at training step t is

$$\eta(t) = \begin{cases} \eta_i + \frac{\eta_{i+1} - \eta_i}{t_{i+1} - t_i}(t - t_i) & \text{if } t_i \leq t < t_{i+1}, \\ \eta_k & \text{if } t \geq t_k. \end{cases}$$

The schedules used by both He et al. (2016a) (piecewise constant) and Goyal et al. (2017) (linear warm-up followed by piecewise constant) for ResNet-50 on ImageNet can both be expressed as piecewise linear.

We ran experiments with ResNet-50 on ImageNet, using Nesterov momentum with batch size 1,024 for 150,000 training steps, while tuning the momentum and all metaparameters governing the learning rate schedule. We used quasi-random metaparameter search as described in Section 4. For piecewise linear schedules, we tried 1, 3, and 5 decay events. We found that it was possible to get good results with several of the schedules we tried, and it is likely that other schedules would also work well. Ultimately, we chose linear decay because it performed at least well as all other schedules we tried, while also being the simplest and requiring only two additional metaparameters.

Appendix D. Additional Plots

(a) Simple CNN on MNIST (b) Transformer on LM1B (c) ResNet-50 on ImageNet
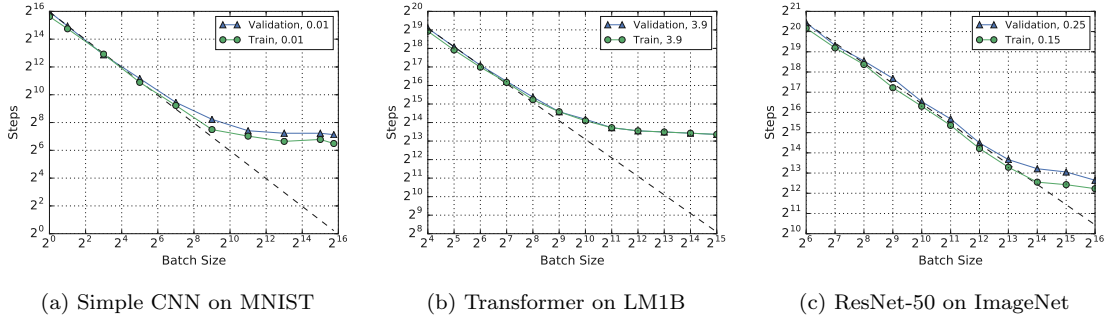
Figure 12: **Steps to result on the training set is almost the same as on the validation set.** The evaluation metrics are described in Appendix A.2. Error goals are specified in the plot legends.
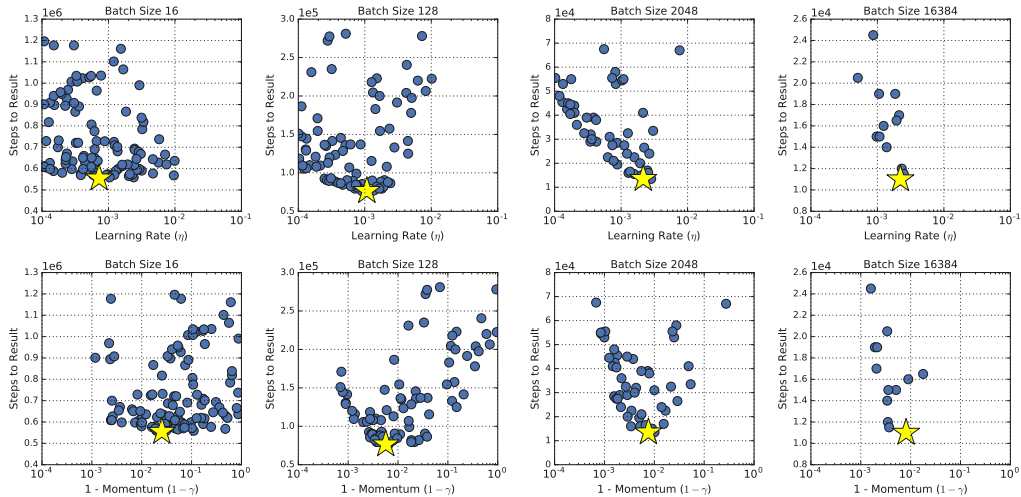

Figure 13: **Validating metaparameter search spaces for Transformer on LM1B.** Rows correspond to the metaparameters we tuned (learning rate η and momentum γ) and columns correspond to different batch sizes. The x-axis is the search range that was sampled by the quasi-random search algorithm. Blue dots represent trials that reached the goal of 3.9 validation cross entropy error, and yellow stars correspond to trials that achieved the goal in the fewest steps. We deem these search spaces appropriate because the yellow stars are not on the boundaries.

37
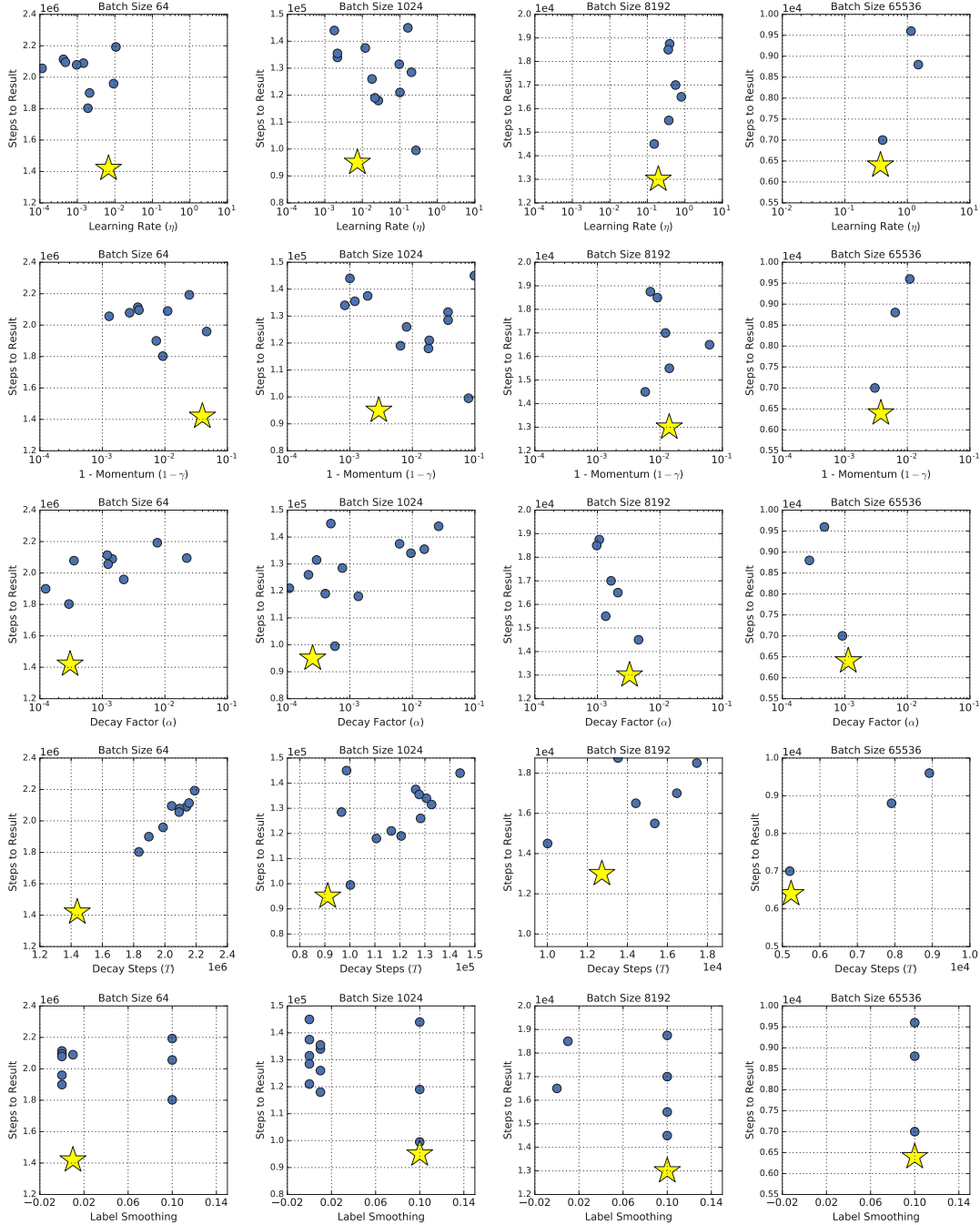
Figure 14: **Validating metaparameter search spaces for ResNet-50 on ImageNet.** Rows correspond to the metaparameters we tuned (initial learning rate η_0, momentum γ, learning rate decay parameters α, T, and label smoothing parameter) and columns correspond to different batch sizes. For all parameters except the label smoothing parameter, the x-axis is the search range sampled by the quasi-random search algorithm. The label smoothing parameter was sampled uniformly in $\{0, 0.01, 0.1\}$ for $b \leq 2^{14}$ and $\{0, 0.1\}$ for $b > 2^{14}$. Blue dots represent trials that reached the goal validation error rate of 0.25, and yellow stars correspond to trials that achieved the goal in the fewest steps. We deem these search spaces appropriate because the yellow stars are not on the boundaries.
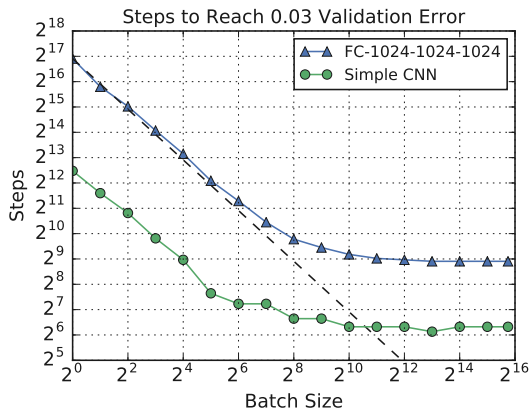
(a) Fully Connected vs Simple CNN on MNIST
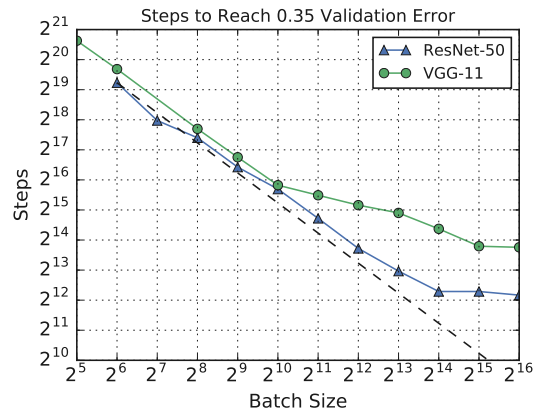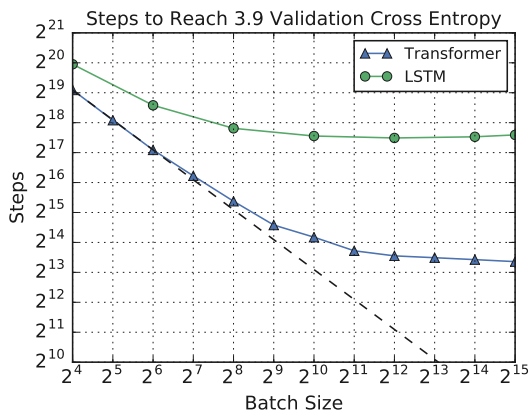
(b) ResNet-50 vs VGG-11 on ImageNet

(c) Transformer vs LSTM on LM1B
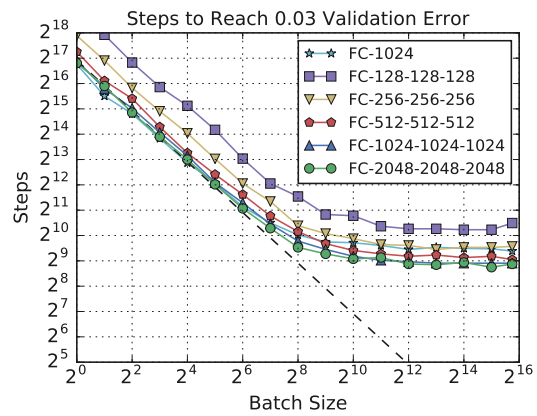
(d) Fully Connected sizes on MNIST
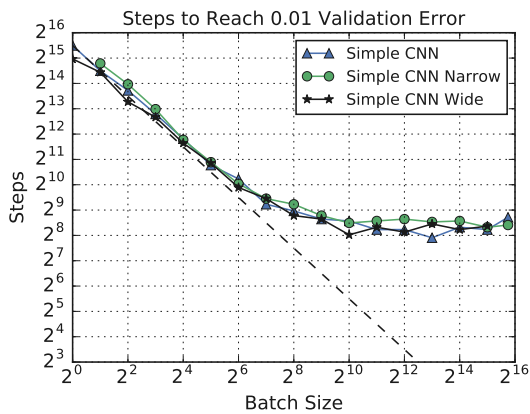
(e) Simple CNN sizes on MNIST
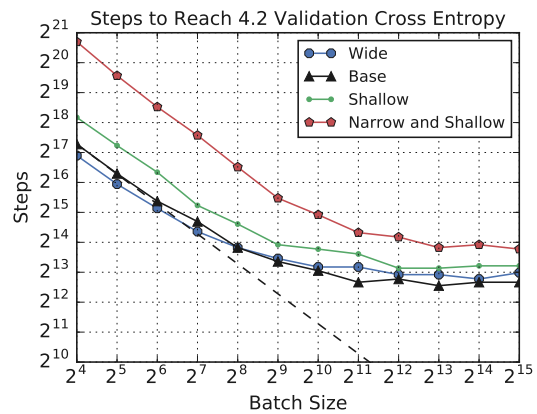
(f) Transformer sizes on LM1B

Figure 15: **Figure 3 without the *y*-axis normalized.**
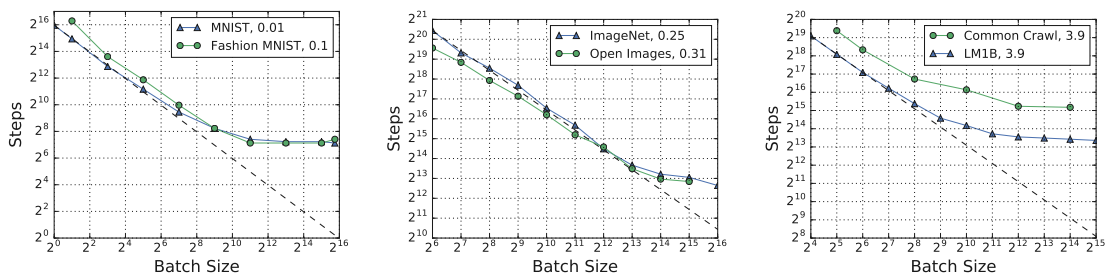
(a) Simple CNN on different data sets   (b) ResNet-50 on different data sets   (c) Transformer on different data sets

Figure 16: **Figure 5 without the y-axis normalized.**
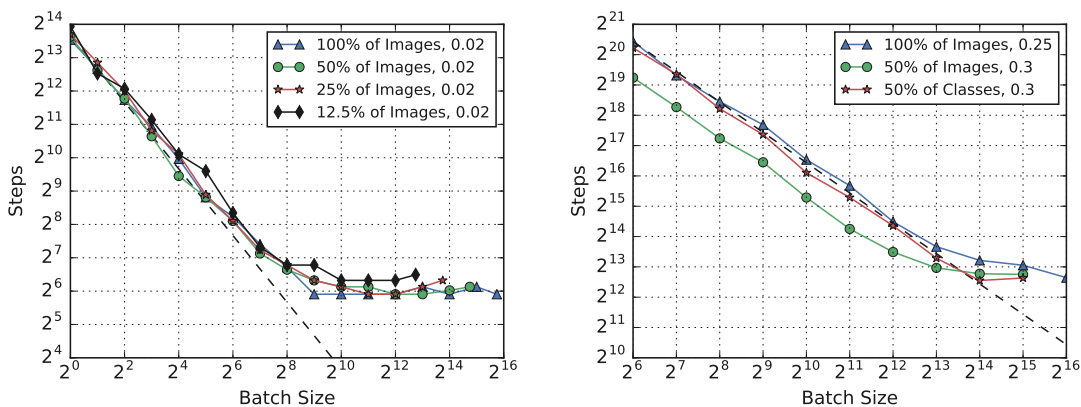
(a) Simple CNN on MNIST subsets   (b) ResNet-50 on ImageNet subsets

Figure 17: **Figure 6 without the y-axis normalized.**
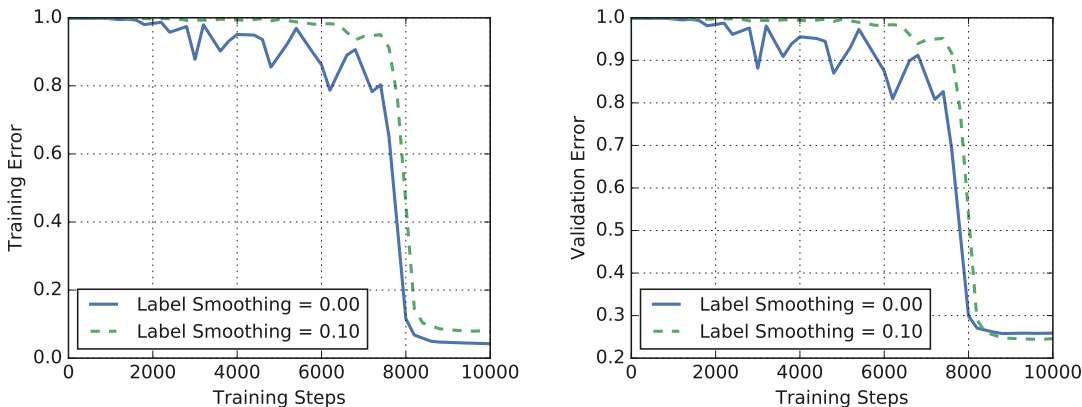
Figure 18: **Label smoothing reduces overfitting at large batch sizes.** Plots are training curves for the two best models with and without label smoothing for ResNet-50 on ImageNet with batch size 2^{16}. The two models correspond to different metaparameter tuning trials, so the learning rate, Nesterov momentum, and learning rate schedule were independently chosen for each trial. The two trials shown are those that reached the highest validation error at any point during training, for label smoothing equal to 0 and 0.1 respectively.

(a) Simple CNN on MNIST   (b) Simple CNN on Fashion MNIST
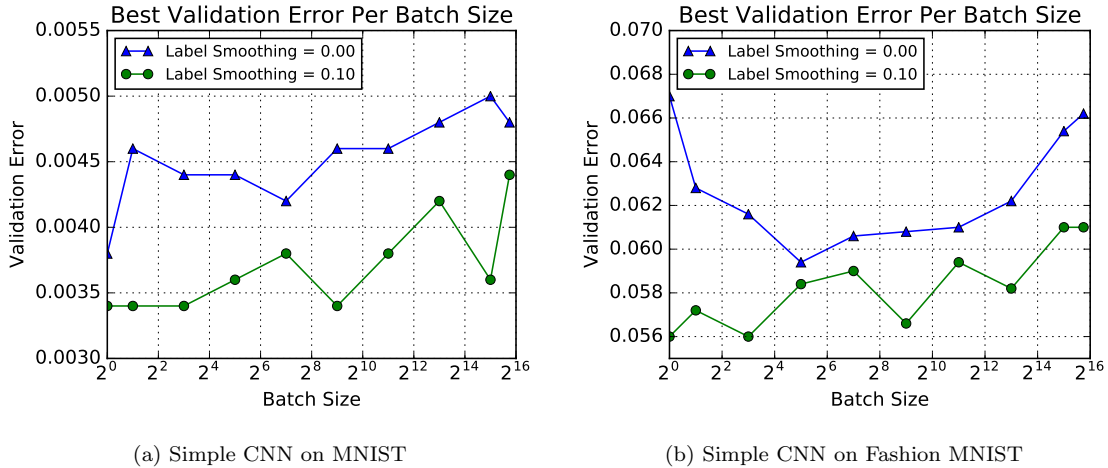
Figure 19: **Label smoothing helps all batch sizes for Simple CNN on MNIST and Fashion MNIST.** There is no consistent trend of label smoothing helping smaller or larger batch sizes more. Each point corresponds to a different metaparameter tuning trial, so the learning rate, Nesterov momentum, and learning rate schedule are independently chosen for each point. The training budget is fixed for each batch size, but varies between batch sizes.
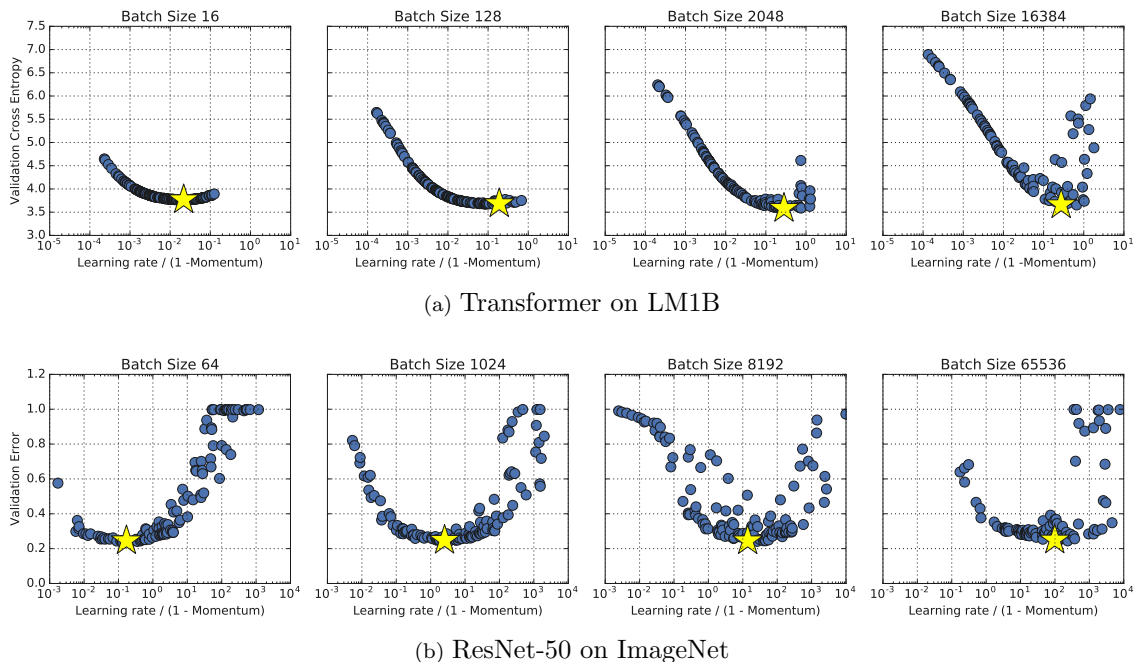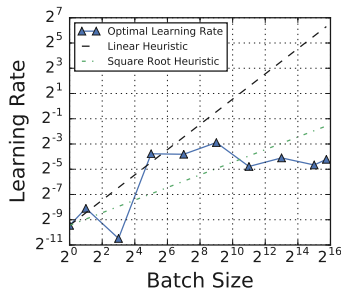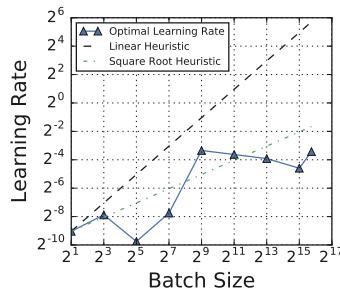

(a) Transformer on LM1B

(b) ResNet-50 on ImageNet

Figure 20: **Validation error vs effective learning rate.** Training budgets are consistent for each batch size, but not between batch sizes. These plots are projections of the entire metaparameter search space, which is 2-dimensional for Transformer on LM1B (see Figure 13) and 5-dimensional for ResNet-50 on ImageNet (see Figure 14).
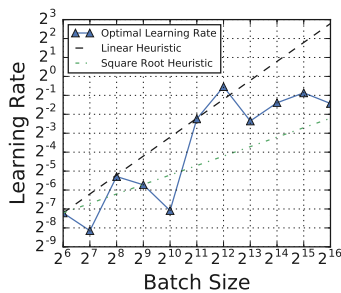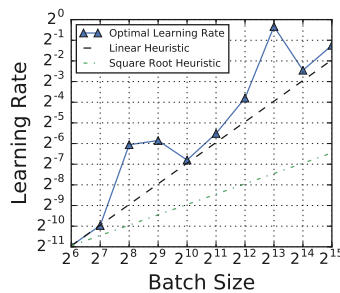
(a) Simple CNN on MNIST
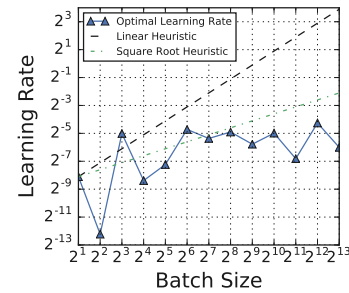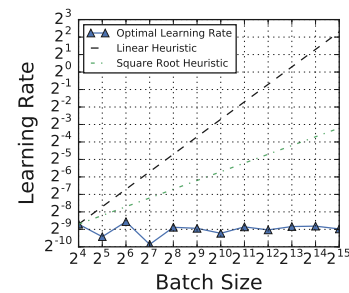
(b) Simple CNN on Fashion MNIST
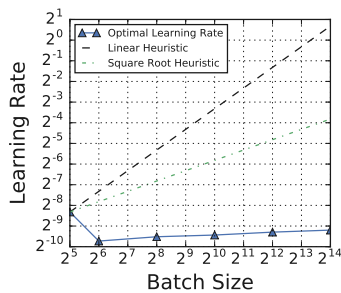
(c) ResNet-8 on CIFAR-10
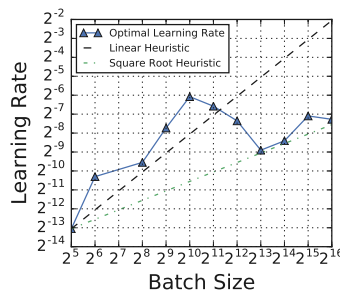
(d) ResNet-50 on ImageNet

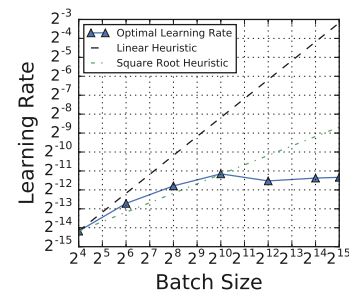(e) ResNet-50 on Open Images

(f) Transformer on LM1B

(g) Transformer on Common Crawl

(h) VGG-11 on ImageNet

(i) LSTM on LM1B

Figure 21: **Optimal learning rates do not always follow linear or square root scaling heuristics.** Learning rates correspond to the trial that reached the goal validation error in the fewest training steps (see Figure 1). For models using learning rate decay schedules (ResNet-8, ResNet-50, VGG-11), plots are based on the initial learning rate. See Figure 22 for the corresponding plot of optimal momentum, and Figure 8 for the corresponding plot of effective learning rate.

(a) Simple CNN on MNIST

(b) Simple CNN on Fashion MNIST

(c) ResNet-8 on CIFAR-10

(d) ResNet-50 on ImageNet

(e) ResNet-50 on Open Images

(f) Transformer on LM1B

(g) Transformer on Common Crawl

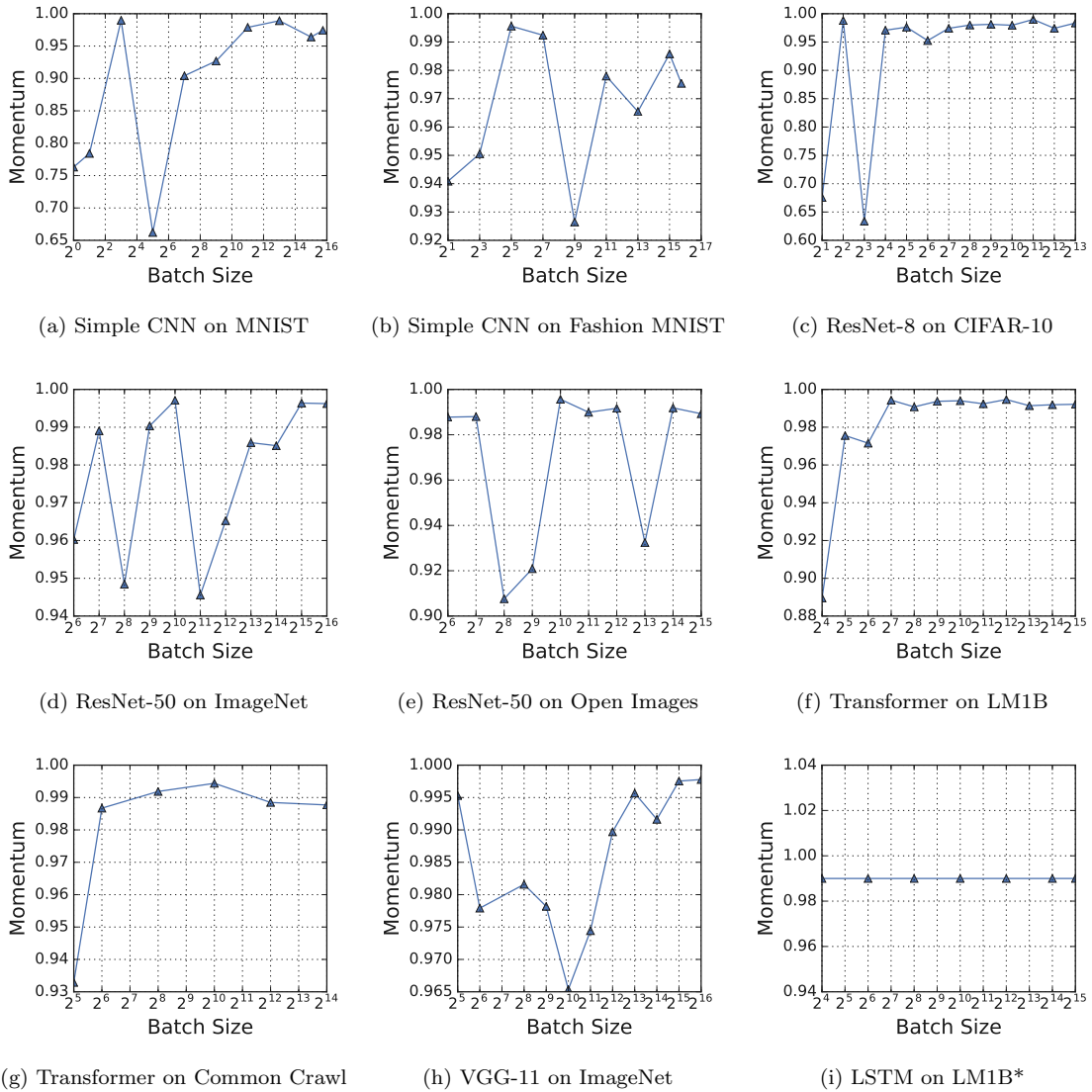(h) VGG-11 on ImageNet

(i) LSTM on LM1B*

Figure 22: **Optimal momentum has no consistent relationship with batch size.** Momentum corresponds to the trial that reached the goal validation error in the fewest training steps (see Figure 1). See Figure 21 for the corresponding plot of optimal learning rate, and Figure 8 for the corresponding plot of effective learning rate. *For LSTM on LM1B, we only tuned η with fixed $\gamma = 0.99$.

43

Step budget   Epoch budget

(a) Simple CNN on MNIST: Validation Error

(b) Simple CNN on MNIST: Test Error
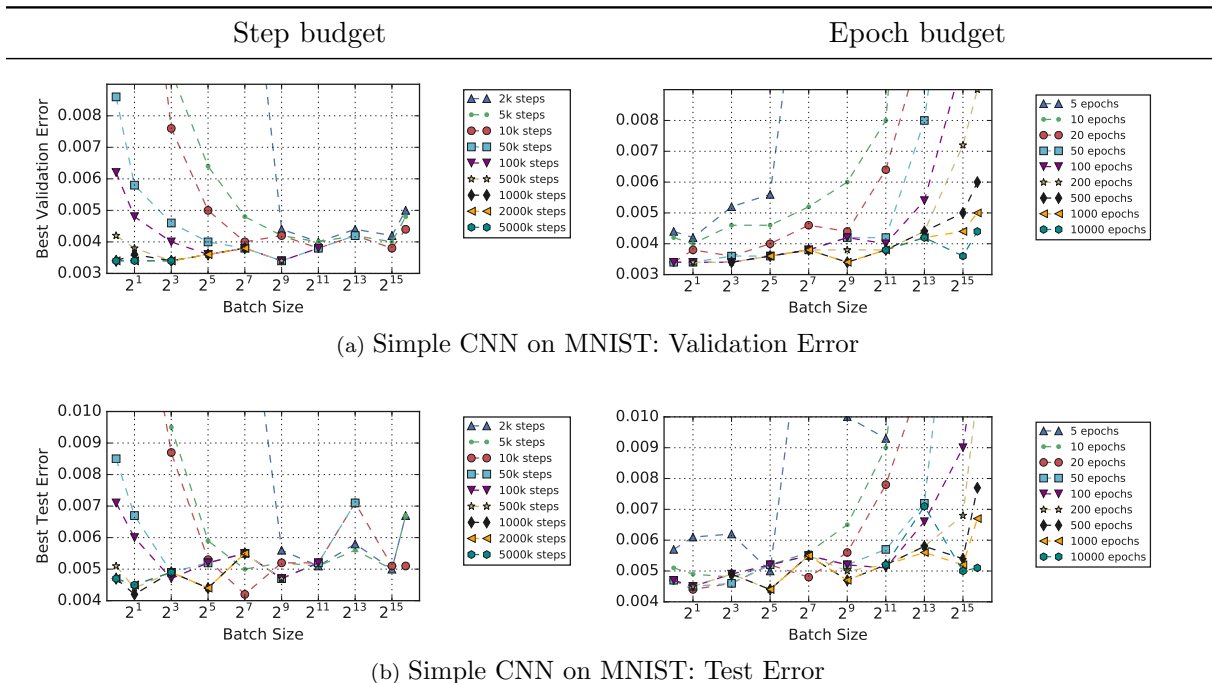
Figure 23: **Zoomed version of Figure 11a.**

Step budget   Epoch budget

(a) Simple CNN on Fashion MNIST: Validation Error

(b) Simple CNN on Fashion MNIST: Test Error
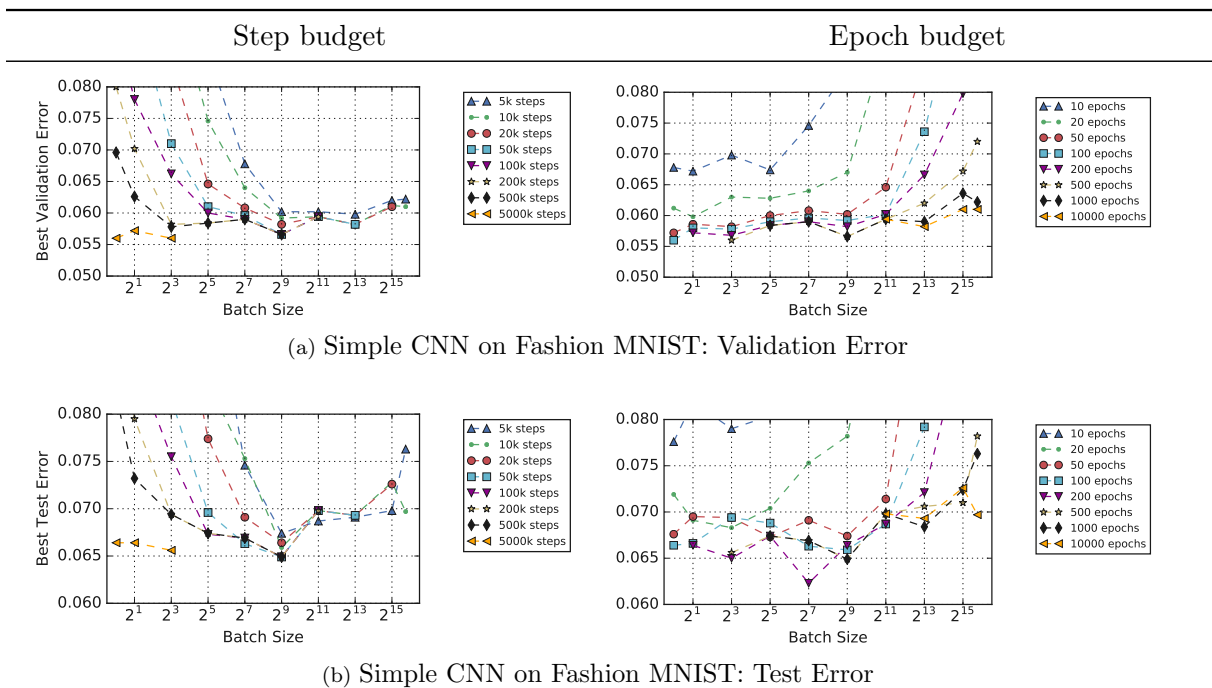
Figure 24: **Zoomed version of Figure 11b.**

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for large-scale machine learning. In *Conference on Operating Systems Design and Implementation*, volume 16, pages 265–283. USENIX, 2016.

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkr1UDeC-.

Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using Kronecker-factored approximations. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SkkTMpjex.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168, 2008.

Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical hyper-parameters: No random, no cry. *arXiv preprint arXiv:1706.03200*, 2017.

Thomas M Breuel. Benchmarking of LSTM networks. *arXiv preprint arXiv:1508.02774*, 2015a.

Thomas M Breuel. The effects of hyperparameters on SGD training of neural networks. *arXiv preprint arXiv:1508.02788*, 2015b.

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. In *Conference of the International Speech Communication Association*, 2014.

Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. In *International Conference on Learning Representations Workshop Track*, 2016. URL https://openreview.net/forum?id=D1VDZ5kMAu5jEJ1zfEWL.

Lingjiao Chen, Hongyi Wang, Jinman Zhao, Dimitris Papailiopoulos, and Paraschos Koutris. The effect of network width on the performance of large-batch training. *arXiv preprint arXiv:1806.03791*, 2018.

Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

Aditya Devarakonda, Maxim Naumov, and Michael Garland. AdaBatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028, 2017.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning, lecture 6a: overview of mini-batch gradient descent, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Prateek Jain, Sham M. Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Parallelizing stochastic gradient descent for least squares regression: Mini-batching, averaging, and model misspecification. *Journal of Machine Learning Research*, 18(223):1–42, 2018. URL `http://jmlr.org/papers/v18/16-595.html`.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture*, pages 1–12. IEEE, 2017.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of Fisher information in deep neural networks: Mean field approach. *arXiv preprint arXiv:1806.01316*, 2018.

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. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=H1oyRlYgg`.

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJTutzbA-`.

Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. OpenImages: A public dataset for large-scale multi-label and multi-class image classification., 2017. URL `https://storage.googleapis.com/openimages/web/index.html`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL `http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1-2):365–397, 2012.

Yann Le Cun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998. URL `http://leon.bottou.org/papers/lecun-98x`.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database, 1998. URL `http://yann.lecun.com/exdb/mnist`.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *International Conference on Knowledge Discovery and Data Mining*, pages 661–670. ACM, 2014.

Tao Lin, Sebastian U Stich, and Martin Jaggi. Don't use large mini-batches, use local SGD. *arXiv preprint arXiv:1808.07217*, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Skq89Scxx`.

Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3331–3340, 2018.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From foundations to algorithms.* Cambridge University Press, 2014. URL `https://books.google.com/books?id=OE9etAEACAAJ`.

Ohad Shamir. Without-replacement sampling for stochastic gradient methods. In *Advances in Neural Information Processing Systems*, pages 46–54, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Samuel L. Smith and Quoc V. Le. A Bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJij4yg0Z`.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *Conference on Computer Vision and Pattern Recognition*, pages 2818–2826. IEEE, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge J Belongie. Learning from noisy large-scale datasets with minimal supervision. In *Conference on Computer Vision and Pattern Recognition*, pages 6575–6583. IEEE, 2017.

D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1MczcgR-`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics*, 2018. URL `http://proceedings.mlr.press/v84/yin18a.html`.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. ImageNet training in minutes. *arXiv preprint arXiv:1709.05011*, 2017.