AdaScale SGD: A User-Friendly Algorithm for Distributed Training

Tyler B. Johnson [† 1] **Pulkit Agrawal** [† 1] **Haijie Gu** [1] **Carlos Guestrin** [1]

Abstract

When using large-batch training to speed up stochastic gradient descent, learning rates must adapt to new batch sizes in order to maximize speed-ups and preserve model quality. Re-tuning learning rates is resource intensive, while fixed scaling rules often degrade model quality. We propose AdaScale SGD, an algorithm that reliably adapts learning rates to large-batch training. By continually adapting to the gradient's variance, AdaScale automatically achieves speed-ups for a wide range of batch sizes. We formally describe this quality with AdaScale's convergence bound, which maintains final objective values, even as batch sizes grow large and the number of iterations decreases. In empirical comparisons, AdaScale trains well beyond the batch size limits of popular "linear learning rate scaling" rules. This includes large-batch training with no model degradation for machine translation, image classification, object detection, and speech recognition tasks. AdaScale's qualitative behavior is similar to that of "warm-up" heuristics, but unlike warm-up, this behavior emerges naturally from a principled mechanism. The algorithm introduces negligible computational overhead and no new hyperparameters, making AdaScale an attractive choice for large-scale training in practice.

1. Introduction

Large datasets and large models underlie much of the recent success of machine learning. Training such models is time consuming, however, often requiring days or even weeks. Faster training enables consideration of more data and models, which expands the capabilities of machine learning.

Stochastic gradient descent and its variants are fundamental training algorithms. During each iteration, SGD applies a small and noisy update to the model, based on a stochastic gradient. By applying thousands of such updates in sequence, a powerful model can emerge over time.

To speed up SGD, a general strategy is to improve the signal-to-noise ratio of each update, i.e., reduce the variance of the stochastic gradient. Some tools for this include SVRG-type gradient estimators (Johnson & Zhang, 2013; Defazio et al., 2014) and prioritization of training data via importance sampling (Needell et al., 2014; Zhao & Zhang, 2015). Data parallelism—our focus in this work—is perhaps the most powerful of such tools. By processing thousands of training examples for each gradient estimate, distributed systems can drastically lower the gradient's variance. Only the system's scalability, not the algorithm, limit the variance reduction.

Smaller variances alone, however, typically result in unimpressive speed-ups. To fully exploit the noise reduction, SGD must also make larger updates, i.e., increase its *learning rate* parameter. While this fact applies universally (including for SVRG (Johnson & Zhang, 2013), data parallelism (Goyal et al., 2017), and importance sampling (Johnson & Guestrin, 2018)), the precise relationship between variance and learning rates remains poorly quantified.

For this reason, practitioners often turn to heuristics in order to adapt learning rates. This applies especially to distributed training, for which case "fixed scaling rules" are standard but unreliable strategies. For example, Goyal et al. (2017) popularized "linear learning rate scaling," which succeeds in limited cases (Krizhevsky, 2014; Devarakonda et al., 2017; Jastrzębski et al., 2018; Smith et al., 2018; Lin et al., 2019). For other problems or greater scales, however, linear scaling leads to poor model quality and even divergence—a result known both in theory (Yin et al., 2018; Jain et al., 2018; Ma et al., 2018) and in practice (Goyal et al., 2017).

In this work, we introduce AdaScale SGD, an algorithm that more reliably adjusts learning rates for large-batch training. AdaScale achieves speed-ups that depend naturally on the gradient's variance before it is decreased—nearly perfect linear speed-ups in cases of large variance, and smaller speed-ups otherwise. For contrast, we also show that as batch sizes increase, linear learning rate scaling progressively degrades model quality, even when including "warm-up" heuristics (Goyal et al., 2017) and additional training iterations.

[†]Equal contribution. [1]Apple, Seattle, WA. Correspondence to: T. Johnson <tbjohns@apple.com>, P. Agrawal <pulkit_agrawal@apple.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

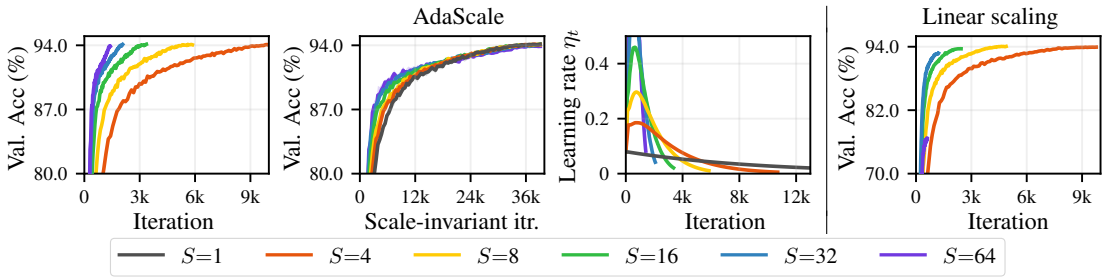

Figure 1: Motivating results. For `cifar10`, AdaScale automatically adapts to many scales S (the number of parallel batches), preserving model quality. When plotted in terms of "scale-invariant" iterations, training curves align closely. With AdaScale, warm-up behavior emerges naturally when adapting a simple learning rate schedule (exponential decay) to scale S (plot cropped to show behavior). Meanwhile, linear scaling (with warm-up) degrades model quality as batch sizes increase.

AdaScale makes large-batch training significantly more user-friendly. Without changing the algorithm's inputs (such as learning rate schedule), AdaScale can adapt to vastly different batch sizes with large speed-ups and near-identical model quality. This has two important implications: (i) AdaScale improves the translation of learning rates to greater amounts of parallelism, which simplifies scaling up tasks or adapting to dynamic resource availability; and (ii) AdaScale works with simple learning rate schedules at scale, eliminating the need for warm-up. Qualitatively, AdaScale and warm-up have similar effects on learning rates, but with AdaScale, the behavior emerges from a principled and adaptive strategy, not hand-tuned parameters.

We perform large-scale empirical evaluations on five training benchmarks. Tasks include image classification, machine translation, object detection, and speech recognition. The results align remarkably well with our theory, as AdaScale systematically preserves model quality across many batch sizes. This includes training ImageNet with batch size 32k and Transformer with 262k max tokens per batch.

The ideas behind AdaScale are not limited to distributed training, but instead apply to any estimator that reduces the gradient's variance. To provide context for the remaining sections, Figure 1 includes results from an experiment using CIFAR-10 data. These results illustrate AdaScale's scaling behavior, the qualitative impact on learning rates, and a failure case for the linear scaling rule.

2. Problem formulation

We focus on quickly and approximately solving

$$\text{minimize}_{\mathbf{w}\in\mathbb{R}^d} \; F(\mathbf{w}) := \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[f(\mathbf{w},\mathbf{x})\right] . \qquad (1)$$

Here $\mathbf{w}$ parameterizes a machine learning model, while $\mathcal{X}$ denotes a distribution over batches of training data. We assume that F and f are differentiable and that $\mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\nabla_{\mathbf{w}} f(\mathbf{w},\mathbf{x})\right] = \nabla F(\mathbf{w})$.

SGD is a fundamental algorithm for solving Problem 1. Let $\mathbf{w}_t$ denote the model parameters when iteration t begins. SGD samples a batch $\mathbf{x}_t \sim \mathcal{X}$ and computes the gradient $\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}} f(\mathbf{w}_t, \mathbf{x}_t)$, before applying the update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{g}_t \, . \qquad (2)$$

Here η_t is the *learning rate*. Given a learning rate schedule $\texttt{lr} : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{>0}$, SGD defines $\eta_t = \texttt{lr}(t)$. For our experiments in §4, `lr` is an exponential decay or step decay function. SGD completes training after T iterations.

To speed up SGD, we can process multiple batches in parallel. Algorithm 1 defines a "scaled SGD" algorithm. At scale S, we sample S independent batches per iteration. After computing the gradient for each batch in parallel, the algorithm synchronously applies the mean of these gradients (in place of $\mathbf{g}_t$ in (2)) when updating the model.

As S increases, scaled SGD generally requires fewer iterations to train a model. But how much speed-up should we expect, especially as S becomes large? Moreover, how do we adapt learning rates in order to realize this speed-up? Practitioners usually answer these questions with one or two approaches: (i) trial-and-error parameter tuning, which requires significant time and specialized knowledge, or (ii) fixed scaling rules, which work well for some problems, but result in poor model quality for many others.

3. AdaScale SGD algorithm

In this section, we introduce AdaScale. We can interpret AdaScale as a per-iteration interpolation between two simple rules for scaling the learning rate. For each of these rules, we first consider an extreme problem setting for which the rule behaves ideally. With this understanding, we then define AdaScale and provide theoretical results. Finally, we discuss approximations needed for a practical implementation.

3.1. Intuition from extreme cases of gradient variance

To help introduce AdaScale, we first consider two simple "scaling rules." A scaling rule translates training parameters

Algorithm 1 Scaled SGD

function `Scaled_SGD`(S, `lr`, T, $\mathcal{X}$, f, $\mathbf{w}_0$)
  for $t = 0, 1, 2, \ldots, T-1$ **do**
    $\bar{\mathbf{g}}_t \leftarrow$ `compute_gradient`($\mathbf{w}_t, S, \mathcal{X}, f$)
    $\eta_t \leftarrow \texttt{lr}(t)$
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \bar{\mathbf{g}}_t$
  return $\mathbf{w}_T$

function `compute_gradient`($\mathbf{w}_t$, S, $\mathcal{X}$, f)
  in parallel for $i = 1, \ldots, S$ **do**
    $\mathbf{x}^{(i)} \leftarrow$ `sample_batch`($\mathcal{X}$)
    $\mathbf{g}^{(i)} \leftarrow \nabla_{\mathbf{w}} f(\mathbf{w}_t, \mathbf{x}^{(i)})$
  return $\frac{1}{S}\sum_{i=1}^{S} \mathbf{g}^{(i)}$

Algorithm 2 AdaScale SGD

function `AdaScale`(S, `lr`, T_{SI}, $\mathcal{X}$, f, $\mathbf{w}_0$)
  initialize $\tau_0 \leftarrow 0$; $t \leftarrow 0$
  while $\tau_t < T_{\text{SI}}$ **do**
    $\bar{\mathbf{g}}_t \leftarrow$ `compute_gradient`($\mathbf{w}_t, S, \mathcal{X}, f$)
    # Compute "gain" $r_t \in [1, S]$ (see (3) and §3.4):
    $r_t \leftarrow \frac{\mathbb{E}_{\mathbf{w}_t}\left[\sigma^2_{\mathbf{g}}(\mathbf{w}_t)+\mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}{\mathbb{E}_{\mathbf{w}_t}\left[\frac{1}{S}\sigma^2_{\mathbf{g}}(\mathbf{w}_t)+\mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}$
    $\eta_t \leftarrow r_t \cdot \texttt{lr}(\lfloor \tau_t \rfloor)$
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \bar{\mathbf{g}}_t$
    $\tau_{t+1} \leftarrow \tau_t + r_t$; $t \leftarrow t+1$
  return $\mathbf{w}_t$

(such as learning rates) to large-batch settings:

Definition 1. *Consider a learning rate schedule* $\texttt{lr}_1$ *and total steps* T_1 *for single-batch training (i.e., Algorithm 1 with* $S = 1$*). Given a scale* S*, a* ***scaling rule*** *maps* $(S, \texttt{lr}_1, T_1)$ *to parameters* $(\texttt{lr}_S, T_S)$ *for training at scale* S*.*

One scaling rule is identity scaling, which keeps training parameters unchanged for all S:

Definition 2. *The* ***identity scaling rule*** *defines* $T_S = T_1$ *and* $\texttt{lr}_S = \texttt{lr}_1$*.*

Note that this rule has little practical appeal, since it fails to reduce the number of training iterations. A more popular scaling rule is linear learning rate scaling:

Definition 3. *The* ***linear scaling rule*** *defines* $T_S = \lceil T_1/S \rceil$ *and* $\texttt{lr}_S(t) = S \cdot \texttt{lr}_1(St)$*.*

Conceptually, linear scaling treats SGD as a perfectly parallelizable algorithm. If true, applying updates from S batches in parallel achieves the same result as doing so in sequence.

For separate special cases of Problem 1, the identity and linear scaling rules maintain training effectiveness for all S. To show this, we first define the gradient variance quantities

$$\boldsymbol{\Sigma}_{\mathbf{g}}(\mathbf{w}) = \text{cov}_{\mathbf{x}\sim\mathcal{X}}(\nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}), \nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}))\,,$$
$$\text{and} \quad \sigma^2_{\mathbf{g}}(\mathbf{w}) = \text{tr}(\boldsymbol{\Sigma}_{\mathbf{g}}(\mathbf{w}))\,.$$

In words, $\sigma^2_{\mathbf{g}}(\mathbf{w})$ sums the variances of each entry in $\nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x})$. Data parallelism fundamentally impacts SGD by reducing this variance.

We first consider the case of deterministic gradients, i.e., $\sigma^2_{\mathbf{g}}(\mathbf{w}) = 0$. Here identity scaling preserves model quality:

Proposition 1 (Identity scaling for zero variance)**.** *Let* $\mathbf{w}^{(1)}$ *denote the result of single batch training, and let* $\mathbf{w}^{(S)}$ *denote the result of scaled training after identity scaling. If* $\sigma^2_{\mathbf{g}}(\mathbf{w}) = 0$ *for all* $\mathbf{w} \in \mathbb{R}^d$*, then* $F(\mathbf{w}^{(1)}) = F(\mathbf{w}^{(S)})$*.*

Although identity scaling does not speed up training, Proposition 1 is important for framing the impact of increasing batch sizes. If the gradient variance is "small," then we cannot expect large gains from scaling up training—further reducing the variance has little effect on $\bar{\mathbf{g}}_t$. With "large" variance, however, the opposite is true:

Proposition 2 (Linear scaling for large variance)**.** *Consider any learning rate* η *and training duration* T*. For some fixed covariance matrix* $\boldsymbol{\Sigma} \in \mathbb{S}^d_{\succ 0}$ *and* $\nu \in \mathbb{Z}_{>0}$*, assume* $\nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}) \sim \mathcal{N}(\nabla F(\mathbf{w}), \nu\boldsymbol{\Sigma})$*. Let* $\texttt{lr}_1(t) = \nu^{-1}\eta$ *and* $T_1 = \nu T$*. Define* $\mathbf{w}^{(1)}$ *as the result of single batch training and* $\mathbf{w}^{(S)}$ *as the result of scaled training after linear scaling. Then* $\mathbb{E}\left[F(\mathbf{w}^{(1)})\right] = \mathbb{E}\left[F(\mathbf{w}^{(S)})\right]$ *in the limit* $\nu \to +\infty$*.*

In simpler terms, linear scaling leads to perfect linear speed-ups in the case of very large gradient variance (as well as small learning rates and many iterations, to compensate for this variance). Since increasing S decreases variance, it is natural that scaling yields large speed-ups in this case.

In practice, the gradient's variance is neither zero nor infinite, and both identity and linear scaling may perform poorly. Moreover, the gradient's variance does not remain constant throughout training. A practical algorithm, it seems, must continually adapt to the state of training.

3.2. AdaScale definition

AdaScale, defined in Algorithm 2, adaptively interpolates between identity and linear scaling, based on the expectation of the gradient's variance. During iteration t, AdaScale multiplies the learning rate by a "gain ratio" $r_t \in [1, S]$:

$$\eta_t = r_t \cdot \texttt{lr}(\lfloor \tau_t \rfloor)\,.$$

Here we define $\tau_t = \sum_{t'=0}^{t-1} r_{t'}$—the *scale-invariant iteration*. The idea is that iteration t performs the equivalent of r_t single-batch iterations, and τ_t accumulates this progress. AdaScale concludes when $\tau_t \geq T_{\text{SI}}$, where T_{SI} is the "total scale-invariant iterations." Since $r_t \in [1, S]$, AdaScale

requires at least $\lceil T_{\text{SI}}/S \rceil$ and at most T_{SI} iterations. For practical problem settings, this training duration often falls closer to $\lceil T_{\text{SI}}/S \rceil$ than T_{SI} (as we see later in §4).

The identity and linear rules correspond to two special cases of AdaScale. If $r_t = 1$ for all t, the algorithm equates to SGD with identity scaling. Similarly, if $r_t = S$ for all t, we have linear scaling. Thus, §3.1 suggests setting $r_t \approx 1$ when $\sigma^2_{\mathbf{g}}(\mathbf{w}_t)$ is small and $r_t \approx S$ when this variance is large. Introducing the notation $\mu^2_{\mathbf{g}}(\mathbf{w}_t) = \|\nabla F(\mathbf{w}_t)\|^2$, AdaScale achieves this by defining

$$r_t = \frac{\mathbb{E}_{\mathbf{w}_t}\left[\sigma^2_{\mathbf{g}}(\mathbf{w}_t) + \mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}{\mathbb{E}_{\mathbf{w}_t}\left[\frac{1}{S}\sigma^2_{\mathbf{g}}(\mathbf{w}_t) + \mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}\,. \tag{3}$$

Here there are expectations both with respect to the batch distribution $\mathcal{X}$ (as part of the variance term, $\sigma^2_{\mathbf{g}}(\mathbf{w}_t)$) and with respect to the distribution over training trajectories. A practical implementation requires estimating r_t, and we describe a procedure for doing so in §3.4. Interestingly, since $\mathbb{E}_{\mathbf{w}_t}\left[\mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]$ decreases toward zero with progress toward convergence, we find empirically that r_t often gradually increases during training, resulting in a "warm-up" effect on the learning rate (see §4). In addition to this intuitive behavior, (3) has a more principled justification. In particular, this r_t ensures that as S increases, $\eta_t \mathbb{E}[\mu^2_{\mathbf{g}}(\mathbf{w}_t)]$ and $\eta_t{}^2 \mathbb{E}[\|\bar{\mathbf{g}}_t\|^2]$ increase multiplicatively by r_t. This leads to the convergence bound for AdaScale that we next present.

3.3. Theoretical results

We now present convergence bounds that describe the speed-ups from AdaScale. Even as the batch size increases, AdaScale's bound converges to the same objective value. We also include an analogous bound for linear scaling.

Let us define $F^* = \min_{\mathbf{w}} F(\mathbf{w})$. Our analysis requires a few assumptions that are typical of SGD analysis of non-convex problems (see, for example, (Lei et al., 2017; Yuan et al., 2019)):

Assumption 1 (α-Polyak-Łojasiewicz)**.** *For some $\alpha > 0$, $F(\mathbf{w}) - F^* \leq \frac{1}{2\alpha}\|\nabla F(\mathbf{w})\|^2$ for all $\mathbf{w}$.*

Assumption 2 (β-smooth)**.** *For some $\beta > 0$, we have $\|\nabla F(\mathbf{w}) - \nabla F(\mathbf{w}')\| \leq \beta\|\mathbf{w} - \mathbf{w}'\|$ for all $\mathbf{w}$, $\mathbf{w}'$.*

Assumption 3 (Bounded variance)**.** *There exists a $V \geq 0$ such that $\sigma^2_{\mathbf{g}}(\mathbf{w}) \leq V$ for all $\mathbf{w}$.*

We emphasize that we do not assume convexity. The PL condition, which is perhaps the strongest of the assumptions, is proven to hold for some nonlinear neural networks (Charles & Papailiopoulos, 2018).

We consider constant `lr` schedules, which result in simple and instructive bounds. Furthermore, constant learning rate schedules provide reasonable results for many deep learning problems (Sun, 2019). To provide context for the AdaScale bound, we first present a result for single-batch training:

Theorem 1 (Single-batch SGD bound)**.** *Given Assumptions 1, 2, 3 and $\eta \in (0, 2\beta^{-1})$, consider Algorithm 1 with $S = 1$ and $\texttt{lr}(t) = \eta$. Defining $\gamma = \eta\alpha(2 - \eta\beta)$ and $\Delta = \frac{1}{2\gamma}\eta^2\beta V$, we have*

$$\mathbb{E}\left[F(\mathbf{w}_T) - F^*\right] \leq (1-\gamma)^T\left[F(\mathbf{w}_0) - F^*\right] + \Delta\,.$$

The bound describes two important characteristics of the single-batch algorithm. First, the suboptimality converges in expectation to at most Δ. Second, convergence to $\Delta + \epsilon$ requires at most $\lceil \log((F(\mathbf{w}_0) - F^*)\epsilon^{-1}) / \log((1-\gamma)^{-1}) \rceil$ iterations. We note similar bounds exist, under a stronger variance assumption (Karimi et al., 2016; Reddi et al., 2016; De et al., 2017; Yin et al., 2018).

Importantly, our AdaScale bound converges to this same Δ for all practical values of S:

Theorem 2 (AdaScale bound)**.** *Define γ, Δ as in Theorem 1. Given Assumptions 1, 2, 3, $S \leq \gamma^{-1}$, and $\eta \in (0, 2\beta^{-1})$, define T as the total iterations of Algorithm 2 with $\texttt{lr}(t) = \eta$ and scale S. Denoting $\bar{r} = \frac{1}{T}\sum_{t=0}^{T-1} r_t$, we have*

$$\mathbb{E}\left[F(\mathbf{w}_T) - F^*\right] \leq (1-\gamma)^{\bar{r}T}\left[F(\mathbf{w}_0) - F^*\right] + \Delta\,.$$

This bound is remarkably similar to that of Theorem 1. Like single-batch SGD, the expected suboptimality converges to at most Δ, but AdaScale achieves this for many batch sizes. Also, AdaScale reduces the total training iterations by a factor $\bar{r}$, the average gain. That is, AdaScale requires at most $\lceil \bar{r}^{-1} \log((F(\mathbf{w}_0) - F^*)\epsilon^{-1}) / \log((1-\gamma)^{-1}) \rceil$ iterations to converge to objective value $\Delta + \epsilon$.

Finally, we provide an analogous bound for linear scaling:

Theorem 3 (Bound for linear scaling rule)**.** *Define γ and Δ as in Theorem 1. Given $\eta \in (0, 2(S\beta)^{-1})$ and Assumptions 1, 2, 3, consider Algorithm 1 with $\texttt{lr}(t) = S\eta$. Defining $\xi(S) = (2 - \eta\beta)/(2 - S\eta\beta)$ and $F_0 = F(\mathbf{w}_0)$, we have*

$$\mathbb{E}\left[F(\mathbf{w}_T) - F^*\right] \leq \left(1 - \tfrac{\gamma}{\xi(S)}\right)^{ST}\left[F_0 - F^*\right] + \xi(S)\cdot\Delta\,.$$

Note $\xi(S) \geq 1$, and this function increases monotonically with S (until an asymptote at $S = 2(\eta\beta)^{-1}$). Thus, unlike in Theorem 2, the bound converges to a value that increases with S. This means that compared to AdaScale, linear scaling often leads to worse model quality and greater risk of divergence, especially for large S. We observe this behavior throughout our empirical comparisons in §4.

Finally, we note Theorem 2 requires $S \leq \gamma^{-1}$—for reasonably small η, this is similar to the stricter constraint $S \leq (2\eta\alpha)^{-1}$. Meanwhile, Theorem 3 requires that $S \leq 2(\eta\beta)^{-1}$. Compared to the constraint for AdaScale, this constraint is typically much stricter, since $\alpha \leq \beta$ (and $\alpha \ll \beta$ if the Hessian's eigenspectrum contains large outlier

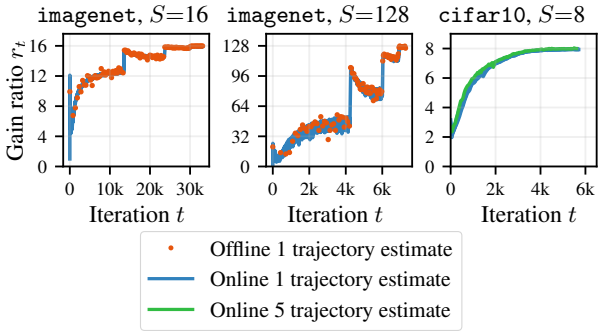

Figure 2: Gain ratios. Plots compare various r_t estimates. In practice, AdaScale uses online estimates from a single training trajectory. We compare to offline estimates (using 1000 batches). We also compare to online estimates that use five independent trajectories (tied only by their shared gain estimates) to average gradient moment estimates across trajectories during each iteration. The values align closely. For `imagenet`, abrupt changes align with `lr` step changes.

values, which has been observed in practice for nerual networks (Sagun et al., 2017; Ghorbani & Krishnan, 2019)). It is also worth noting that if γ is large enough that $S \geq \gamma^{-1}$, then training at smaller scales converges quickly (due to Theorem 1), and large batch sizes are likely unnecessary.

3.4. Practical considerations

A practical AdaScale implementation must efficiently approximate the gain ratio r_t for all iterations. Fortunately, the per-batch gradients $\mathbf{g}_t^{(1)}, \ldots, \mathbf{g}_t^{(S)}$ and aggregated gradient $\bar{\mathbf{g}}_t$ are readily available in distributed SGD algorithms. This makes approximating r_t straightforward, since in addition to (3), we can express r_t as the ratio of expectations

$$r_t = \frac{\mathbb{E}[\frac{1}{S}\sum_{i=1}^{S} \|\mathbf{g}_t^{(i)}\|^2]}{\mathbb{E}[\|\bar{\mathbf{g}}_t\|^2]} \,.$$

Here we take the expectation over all randomness of the algorithm (both current and prior batches).

To estimate the gain, we recommend tracking moving averages of $\frac{1}{S}\sum_{i=1}^{S} \|\mathbf{g}_t^{(i)}\|^2$ and $\|\bar{\mathbf{g}}_t\|^2$ across iterations. Denoting these averages by m_1 and m_2, respectively, we can estimate the gain as $\hat{r}_t = \frac{m_1+\epsilon}{m_2+\epsilon}$. Here ϵ is a small constant, such as 10^{-6}, for numerical stability.

For our empirical results, we use exponential moving averages with parameter $\theta = \max\{1 - S/1000, 0\}$, where $\theta = 0$ results in no averaging. We find that AdaScale is robust to the choice of θ, and we provide evidence of this in supplementary material. When ensuring numerical stability, the implementation for this work also uses an alternative to our recommendation of adding ϵ to m_1 and m_2 (see the supplement for details). Our recommended strategy simplifies gain estimation but should not significantly affect results, since usually $\|\bar{\mathbf{g}}_t\|^2 \gg \epsilon$ in practice.

Figure 2 verifies the usefulness of our estimator by comparing AdaScale's estimates to improved (but impractical) ones. Moving averages (i) add robustness to estimation variance and (ii) incorporate multiple points from the optimization space. For (ii), we ideally would use points from independent training trajectories, but this is impractical. Even so, Figure 2 suggests that estimates from single and multiple trajectories can align closely. We note that numerous prior works—for example, (Schaul et al., 2013; Kingma & Ba, 2015; McCandlish et al., 2018)—have relied on similar moving averages to estimate gradient moments.

One final practical consideration is the momentum parameter ρ when using AdaScale with momentum-SGD. The performance of momentum-SGD depends less critically on ρ than the learning rate (Shallue et al., 2019). For this reason, AdaScale often performs well if ρ remains constant across scales and iterations. This approach to momentum scaling has also succeeded in prior works involving the linear scaling rule (Goyal et al., 2017; Smith et al., 2018).

4. Empirical evaluation

We evaluate AdaScale on five practical training benchmarks. We consider a variety of tasks, models (He et al., 2016a;b; Amodei et al., 2016; Vaswani et al., 2017; Redmon & Farhadi, 2018), and datasets (Deng et al., 2009; Krizhevsky, 2009; Everingham et al., 2010; Panayotov et al., 2015). Table 1 summarizes these training benchmarks. We provide additional implementation details in the supplement.

For each benchmark, we use *one simple learning rate schedule*. Specifically, `lr` is an exponential decay function for `cifar10` and `speech`, and a step decay function otherwise. We use standard `lr` parameters for `imagenet` and `yolo`. Otherwise, we use tuned parameters that approximately maximize the validation metric (to our knowledge,

Table 1: Overview of training benchmarks.

Name	Task	Model	Dataset	Metric
`cifar10`	Image classification	ResNet-18 (v2)	CIFAR-10	Top-1 accuracy (%)
`imagenet`	Image classification	ResNet-50 (v1)	ImageNet	Top-1 accuracy (%)
`speech`	Speech recognition	Deep speech 2	LibriSpeech	Word accuracy (%)
`transformer`	Machine translation	Transformer base	WMT-2014	BLEU
`yolo`	Object detection	YOLOv3	PASCAL VOC	mAP (%)

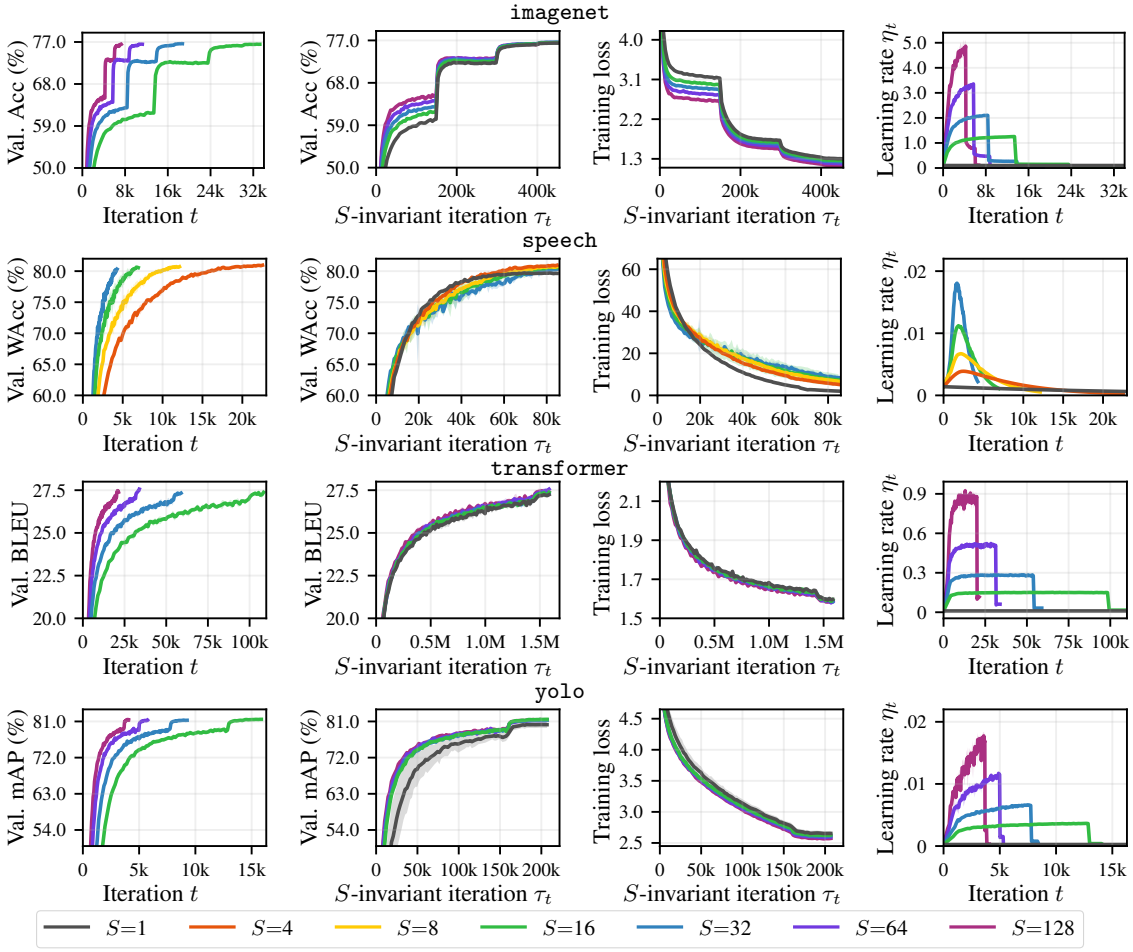

Figure 3: AdaScale training curves. For many scales and benchmarks, AdaScale trains quality models. Training curves align closely in terms of τ_t. In all cases, η_t warms up gradually at the start of training, even though all `lr` schedules are simple exponential or step decay functions (which are non-increasing in t).

there are no standard schedules for solving `speech` and `transformer` with momentum-SGD). We use momentum $\rho = 0.9$ except for `transformer`, in which case we use $\rho = 0.99$ for greater training stability.

Figure 3 (and Figure 1) contains AdaScale training curves for the benchmarks and many scales. Each curve plots the mean of five distributed training runs with varying random seeds. As S increases, AdaScale trains for fewer iterations and consistently preserves model quality. Interestingly, the training curves align closely when plotted in terms of scale-invariant iterations.

For $S > 1$, AdaScale's learning rate increases gradually during initial training, despite the fact that `lr` is non-increasing. Unlike warm-up, this behavior emerges naturally from a principled algorithm, not hand-tuned user input. Thus, AdaScale provides not only a compelling alternative to warm-up but also a plausible explanation for warm-up's success.

For `imagenet`, we also consider elastic scaling. Here, the only change to AdaScale is that S changes abruptly after some iterations. We consider two cases: (i) S increases from 32 to 64 at $\tau_t = T_{\text{SI}}/4$ and from 64 to 128 at $\tau_t = T_{\text{SI}}/2$, and (ii) the scale decreases at the same points, from 128 to 64 to 32. In Figure 4, we include training curves from this setting. Despite the abrupt batch size changes, AdaScale trains quality models, highlighting AdaScale's value for the common scenario of dynamic resource availability.

As a baseline for all benchmarks, we also evaluate linear scaling with warm-up (LSW). As inputs, LSW takes single-batch schedule $\texttt{lr}_1 = \texttt{lr}$ and duration $T_1 = T_{\text{SI}}$, where `lr` and T_{SI} are the inputs to AdaScale. Our warm-up implementation closely follows that of Goyal et al. (2017). LSW trains for $\lceil T_1/S \rceil$ iterations, applying warm-up to the first 5.5% of iterations. During warm-up, the learning rate increases linearly from $\texttt{lr}_1(0)$ to $S \cdot \texttt{lr}_1(0)$.

Since LSW trains for fewer iterations than AdaScale, we also consider a stronger baseline, LSW+, which matches

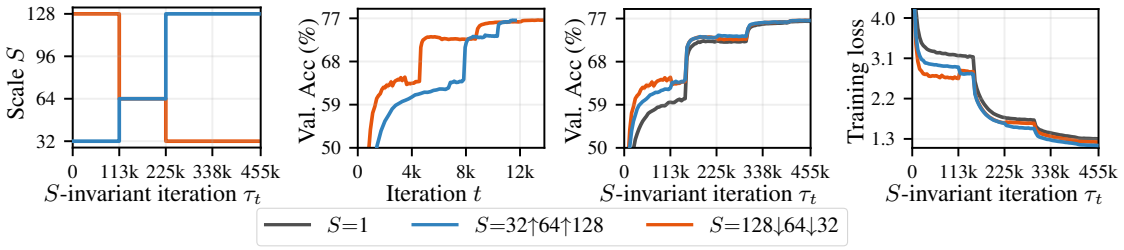

Figure 4: Elastic AdaScaling. For `imagenet`, AdaScale scales training successfully even with abrupt changes to S (at $\tau_t = 133\text{k}, 225\text{k}$). Unlike AdaScale, LSW degrades model quality in this setting (see Table 2).

AdaScale in total iterations. LSW+ uses the same learning rate schedule as LSW except scaled (stretched) along the iterations axis by the difference in training duration. We note LSW+ is *significantly less practical* than LSW and AdaScale, since it requires either (i) first running AdaScale to determine the number of iterations, or (ii) tuning the number of iterations.

Table 2 compares results for AdaScale, LSW, and LSW+. LSW consistently trains for fewer iterations, but doing so comes at a cost. As S grows larger, LSW consistently degrades model quality and sometimes diverges. For these divergent cases, we also tested doubling the warm-up duration to 11% of iterations, and training still diverged. Similarly, even with the benefit of additional iterations, LSW+ also produces worse model quality in many cases. In contrast, AdaScale preserves model quality for nearly all cases.

Table 2: Comparison of final model quality. *Shorthand:* AS=AdaScale, LSW=Linear scaling with warm-up, LSW+=Linear scaling with warm-up and additional steps, gray=model quality significantly worse than for $S = 1$ (5 trials, 0.95 significance), N/A=training diverges, Elastic↑/↓=elastic scaling with increasing/decreasing scale (see Figure 4). Linear scaling leads to poor model quality as the scale increases; AdaScale preserves model performance for nearly all cases.

Task	S	Total batch size	Validation metric			Training loss			Total iterations		
			AS	LSW	LSW+	AS	LSW	LSW+	AS	LSW	LSW+
`cifar10`	1	128	94.1	94.1	94.1	0.157	0.157	0.157	39.1k	39.1k	39.1k
	8	1.02k	94.1	94.0	94.0	0.153	0.161	0.145	5.85k	4.88k	5.85k
	16	2.05k	94.1	93.6	94.1	0.150	0.163	0.136	3.36k	2.44k	3.36k
	32	4.10k	94.1	92.8	94.0	0.145	0.177	0.128	2.08k	1.22k	2.08k
	64	8.19k	93.9	76.6	93.0	0.140	0.272	0.140	1.41k	611	1.41k
`imagenet`	1	256	76.4	76.4	76.4	1.30	1.30	1.30	451k	451k	451k
	16	4.10k	76.5	76.3	76.5	1.26	1.31	1.27	33.2k	28.2k	33.2k
	32	8.19k	76.6	76.1	76.4	1.23	1.33	1.24	18.7k	14.1k	18.7k
	64	16.4k	76.5	75.6	76.5	1.19	1.35	1.20	11.2k	7.04k	11.2k
	128	32.8k	76.5	73.3	75.5	1.14	1.51	1.14	7.29k	3.52k	7.29k
	Elastic↑	various	76.6	75.7	–	1.15	1.37	–	11.6k	7.04k	–
	Elastic↓	various	76.6	74.1	–	1.23	1.45	–	13.6k	9.68k	–
`speech`	1	32	79.6	79.6	79.6	2.03	2.03	2.03	84.8k	84.8k	84.8k
	4	128	81.0	80.9	81.0	5.21	4.66	4.22	22.5k	21.2k	22.5k
	8	256	80.7	80.2	80.7	6.74	6.81	6.61	12.1k	10.6k	12.1k
	16	512	80.6	N/A	N/A	7.33	N/A	N/A	6.95k	5.30k	6.95k
	32	1.02k	80.3	N/A	N/A	8.43	N/A	N/A	4.29k	2.65k	4.29k
`transformer`	1	2.05k	27.2	27.2	27.2	1.60	1.60	1.60	1.55M	1.55M	1.55M
	16	32.8k	27.4	27.3	27.4	1.60	1.60	1.59	108k	99.0k	108k
	32	65.5k	27.3	27.0	27.3	1.59	1.61	1.59	58.9k	49.5k	58.9k
	64	131k	27.6	26.7	27.1	1.59	1.63	1.60	33.9k	24.8k	33.9k
	128	262k	27.4	N/A	N/A	1.59	N/A	N/A	21.4k	12.1k	21.4k
`yolo`	1	16	80.2	80.2	80.2	2.65	2.65	2.65	207k	207k	207k
	16	256	81.5	81.4	81.9	2.63	2.66	2.47	15.9k	12.9k	15.9k
	32	512	81.3	80.5	81.7	2.61	2.81	2.42	9.27k	6.47k	9.27k
	64	1.02k	81.3	70.1	80.6	2.60	4.02	2.51	5.75k	3.23k	5.75k
	128	2.05k	81.4	N/A	N/A	2.57	N/A	N/A	4.07k	1.62k	4.07k

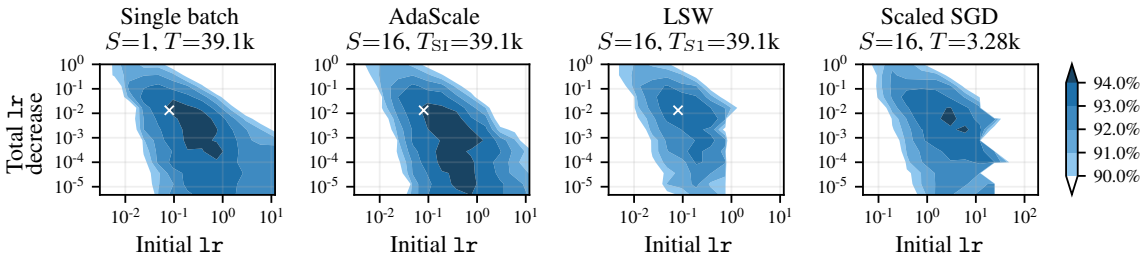

Figure 5: AdaScale results for many learning rate schedules. Heat maps cover the space of exponential decay `lr` schedules for `cifar10`. At scale 16, validation accuracies for AdaScale align closely with results for single-batch training, with the space of 94+% schedules growing moderately with AdaScale. With LSW, no schedule achieves 94% accuracy. On the right, direct `lr` search at scale 16 produces inferior results to AdaScale (here the total iterations, 3.28k, is the average total iterations among 94+% AdaScale trials). Thus, AdaScale induces a superior family of schedules for large-batch training. The white '$\times$' indicates the `lr` used for Figure 1.

As a final comparison, Figure 5 demonstrates AdaScale's performance on `cifar10` with many different `lr` schedules. We consider a 13×13 grid of exponential decay schedules and plot contours of the resulting validation accuracies. At scale 16, AdaScale results align with results for single-batch training, illustrating that AdaScale preserves model quality for many schedules. Moreover, AdaScale convincingly outperforms direct search over exponential decay schedules for scaled SGD at S=16. This suggests that AdaScale provides a better learning rate family for distributed training.

5. Relation to prior work

While linear scaling with warm-up is perhaps the most popular fixed scaling rule, researchers have considered a few alternative strategies. "Square root learning rate scaling" (Krizhevsky, 2014; Li et al., 2014; Hoffer et al., 2017; You et al., 2018) multiplies learning rates by the square root of the batch size increase. Across scales, this preserves the covariance of the SGD update. Establishing this invariant remains poorly justified, however, and often root scaling degrades model quality in practice (Goyal et al., 2017; Golmant et al., 2018; Jastrzębski et al., 2018). AdaScale adapts learning rates by making $\eta_t \mathbb{E}\left[\|\bar{\mathbf{g}}_t\|^2\right]$ invariant across scales, which results in our bound from §3.3. Shallue et al. (2019) compute near-optimal parameters for many tasks and scales, and the results do not align with any fixed rule. To ensure effective training, the authors recommend avoiding such rules and re-tuning parameters for each new scale. This solution is inconvenient and resource-intensive, however, and Shallue et al. do not consider adapting learning rates to the state of training.

Many prior works have also considered the role of gradient variance in SGD. Yin et al. (2018) provide conditions—including sufficiently small batch size and sufficiently large gradient variance—under which linear learning rate scaling works well. Yin et al. do not provide an alternative strategy for adapting learning rates when linear scaling fails. McCandlish et al. (2018) study the impact of gradient variance on scaling efficiency. By averaging the relative gradient variance over the course of training, they make rough (yet fairly accurate) estimates of training time complexities as a function of scale. While McCandlish et al. (2018) do not provide an algorithm for obtaining such speed-ups, these general findings also relate to AdaScale, since gradient variance similarly determines AdaScale's efficiency. Much like AdaScale, Johnson & Guestrin (2018) also adapt learning rates to lower amounts of gradient variance—in this case when using SGD with importance sampling. Because the variance reduction is relatively small in this setting, however, distributed training can have far greater impact on training times. Lastly, many algorithms also adapt to gradient moments for improved training, given a single batch size—see (Schaul et al., 2013; Kingma & Ba, 2015; Balles & Hennig, 2018), just to name a few. In contrast, AdaScale translates learning rates for one batch size into learning rates for a larger scale. Perhaps future versions of AdaScale will combine approaches and achieve both goals. You et al. (2017; 2020) scaled training to large batch sizes by combining adaptive gradient algorithms with scaling rule heuristics.

6. Discussion

SGD is not perfectly parallelizable. Unsurprisingly, the linear scaling rule can fail at large scales. In contrast, AdaScale accepts sublinear speedups in order to better preserve model quality. What do the speed-ups from AdaScale tell us about the scaling efficiency of SGD in general? For many problems, such as `imagenet` with batch size 32.8k, AdaScale provides lower bounds on SGD's scaling efficiency. An important remaining question is whether AdaScale is close to optimally efficient, or if other practical algorithms can achieve similar model quality with fewer iterations.

AdaScale establishes a useful new parameterization of learning rate schedules for large-batch SGD. Practitioners can provide a simple `lr` schedule, which AdaScale adapts to

learning rates for scaled training. From this, warm-up behavior emerges naturally, which produces quality models for many problems and scales. Even in elastic scaling settings, AdaScale adapts successfully to the state of training. Given these appealing qualities, it seems important to further study this family of learning rate schedules.

Based on our empirical results, as well as the algorithm's practicality and theoretical justification, we believe AdaScale can be very valuable for distributed training.

Acknowledgements

For valuable feedback, we thank Emad Soroush, David Dai, Wei Fang, Okan Akalin, Russ Webb, and Kunal Talwar.

References

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J. H., Fan, L., Fougner, C., Han, T., Hannun, A. Y., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A. Y., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J., and Zhu, Z. Deep speech 2: End-to-end speech recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Balles, L. and Hennig, P. Dissecting Adam: The sign, magnitude and variance of stochastic gradients. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Charles, Z. and Papailiopoulos, D. Stability and generalization of learning algorithms that converge to global optima. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

De, S., Yadav, A., Jacobs, D., and Goldstein, T. Automated inference with adaptive batches. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

Defazio, A., Bach, F., and Lacoste-Julien, S. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*, 2014.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Devarakonda, A., Naumov, M., and Garland, M. AdaBatch: Adaptive batch sizes for training deep neural networks. arXiv:1712.02029, 2017.

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88 (2):303–338, 2010.

Ghorbani, B. and Krishnan, S. An investigation into neural net optimization via hessian eigenvalue density. arXiv:1901.10159, 2019.

Golmant, N., Vemuri, N., Yao, Z., Feinberg, V., Gholami, A., Rothauge, K., Mahoney, M. W., and Gonzalez, J. On the computational inefficiency of large batch sizes for stochastic gradient descent. arXiv:1811.12941, 2018.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: Training ImageNet in one hour. arXiv:1706.02677, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, 2016b.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems 30*, 2017.

Jain, P., Kakade, S. M., Kidambi, R., Netrapalli, P., and Sidford, A. Parallelizing stochastic gradient descent for least squares regression: Mini-batching, averaging, and model misspecification. *Journal of Machince Learning Research*, 18(223):1–42, 2018.

Jastrzębski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. J. Three factors influencing minima in SGD. In *Proceedings of the 27th International Conference on Artificial Neural Networks*, 2018.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, 2013.

Johnson, T. B. and Guestrin, C. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems 31*, 2018.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Kloeden, P. E. and Platen, E. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997, 2014.

Lei, L., Ju, C., Chen, J., and Jordan, M. I. Nonconvex finite-sum optimization via SCSG methods. In *Advances in Neural Information Processing Systems 30*, 2017.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD Interational Conference on Knowledge Discovery and Data Mining*, 2014.

Lin, H., Zhang, H., Ma, Y., He, T., Zhang, Z., Zha, S., and Li, M. Dynamic mini-batch SGD for elastic distributed training: Learning in the limbo of resources. arXiv:1904.12043, 2019.

Ma, S., Bassily, R., and Belkin, M. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training. arXiv:1812.06162, 2018.

Needell, D., Ward, R., and Srebro, N. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems 27*, 2014.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015.

Reddi, S. J., Hefny, A., Sra, S., Póczoós, B., and Smola, A. Stochastic variance reduction for nonconvex optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Redmon, J. and Farhadi, A. YOLOv3: An incremental improvement. arXiv:1804.02767, 2018.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. arXiv:1706.04454, 2017.

Schaul, T., Zhang, S., and LeCun, Y. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *Journal of Machince Learning Research*, 20(112):1–49, 2019.

Smith, S., Kindermans, P., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

Sun, R. Optimization for deep learning: theory and algorithms. arXiv:1912.08957, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 31*, 2017.

Yin, D., Pananjady, A., Lam, M., Papailiopoulos, D., Ramchandran, K., and Bartlett, P. Gradient diversity: A key ingredient for scalable distributed learning. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, 2018.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. arXiv:1708.03888, 2017.

You, Y., Hseu, J., Ying, C., Demmel, J., Keutzer, K., and Hsieh, C. Large-batch training for LSTM and beyond. In *NeurIPS Workshop on Systems for ML and Open Source Software*, 2018.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

Yuan, Z., Yan, Y., Jin, R., and Yang, T. Stagewise training accelerates convergence of testing error over sgd. In *Advances in Neural Information Processing Systems 32*, 2019.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. Mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Zhang, Z., He, T., Zhang, H., Zhang, Z., Xie, J., and Li, M. Bag of freebies for training object detection neural networks. arXiv:1902.04103, 2019.

Zhao, P. and Zhang, T. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.