Coupling Adaptive Batch Sizes with Learning Rates

Lukas Balles
MPI for Intelligent Systems
Tübingen, Germany

Javier Romero*
Body Labs Inc.
New York, NY, USA

Philipp Hennig
MPI for Intelligent Systems
Tübingen, Germany

Abstract

Mini-batch stochastic gradient descent and variants thereof have become standard for large-scale empirical risk minimization like the training of neural networks. These methods are usually used with a constant batch size chosen by simple empirical inspection. The batch size significantly influences the behavior of the stochastic optimization algorithm, though, since it determines the variance of the gradient estimates. This variance also changes over the optimization process; when using a constant batch size, stability and convergence is thus often enforced by means of a (manually tuned) decreasing learning rate schedule.

We propose a practical method for dynamic batch size adaptation. It estimates the variance of the stochastic gradients and adapts the batch size to decrease the variance proportionally to the value of the objective function, removing the need for the aforementioned learning rate decrease. In contrast to recent related work, our algorithm couples the batch size to the learning rate, directly reflecting the known relationship between the two. On popular image classification benchmarks, our batch size adaptation yields faster optimization convergence, while simultaneously simplifying learning rate tuning. A TensorFlow implementation is available.

1 INTRODUCTION

In parametric machine learning models, like logistic regression or neural networks, the performance of a parameter vector $w \in \mathbb{R}^d$ on datum x is quantified by a loss function $\ell(w; x)$. Assuming the data comes from a distribution $x \sim p$, the goal is to minimize the expected loss, or *risk*,

$$\mathcal{R}(w) = \mathbf{E}_{x\sim p}[\ell(w; x)]. \tag{1}$$

We consider *empirical risk minimization* tasks of the form

$$\min_{w\in\mathbb{R}^d} F(w) = \frac{1}{M}\sum_{i=1}^{M} \ell(w; x_i), \tag{2}$$

where the risk is approximated using a training set $\{x_1, \dots, x_M\}$ of data sampled (approximately) from p. Typical optimization algorithms used to minimize (2) repeatedly evaluate the gradient

$$\nabla F(w) = \frac{1}{M}\sum_{i=1}^{M} \nabla\ell(w; x_i). \tag{3}$$

For large-scale problems where M and/or d are large, it is inefficient or impossible to evaluate the exact gradient (3), and one typically resorts to stochastic gradients by randomly drawing a mini-batch $\mathcal{B} \subset \{1, \dots, M\}$, $|\mathcal{B}| = m \ll M$, at each step of the optimization algorithm and using the gradient approximation

$$g(w) = \frac{1}{m}\sum_{i\in\mathcal{B}} \nabla\ell(w; x_i). \tag{4}$$

The simplest, but still widely used, stochastic optimization algorithm is stochastic gradient descent (SGD, Robbins & Monro, 1951), which updates

$$w_{k+1} = w_k - \alpha_k g(w_k), \tag{5}$$

where $\alpha_k \in \mathbb{R}_+$ is the step size parameter, often called *learning rate* in the machine learning context. Variants of SGD include ADAGRAD (Duchi et al., 2011), ADADELTA (Zeiler, 2012), and ADAM (Kingma & Ba, 2015). We restrict our considerations to SGD in this paper.

* Work done while at MPI for Intelligent Systems.

1.1 THE EFFECT OF THE BATCH SIZE

If i is drawn uniformly at random from $\{1, \dots, M\}$, $\nabla \ell_i(w) = \nabla \ell(w; x_i)$ is a random variable with mean

$$\mathbf{E}[\nabla \ell_i(w)] = \frac{1}{M} \sum_{i=1}^{M} \nabla \ell_i(w) = \nabla F(w), \qquad (6)$$

and covariance matrix

$$\Sigma(w) := \mathbf{cov}\left[\nabla \ell_i(w)\right] = \frac{1}{M} \sum_{i=1}^{M} (\nabla \ell_i(w) - \nabla F(w))(\nabla \ell_i(w) - \nabla F(w))^T. \qquad (7)$$

Likewise, a stochastic gradient $g(w)$ computed on a randomly-drawn mini-batch $\mathcal{B}$ is a random variable with mean $\nabla F(w)$. Assuming that it is composed of m samples drawn independently with replacement, its covariance matrix is

$$\mathbf{cov}[g(w)] = \frac{\Sigma(w)}{m} \qquad (8)$$

and, by the Central Limit Theorem, $g(w)$ is approximately normally distributed:

$$g(w) \sim \mathcal{N}\left(\nabla F(w), \frac{\Sigma(w)}{m}\right). \qquad (9)$$

When sampling without replacement, as is usually done in practice, the same holds *approximately* as long as $m \ll M$.

In practice, the batch size m is often set to a fixed value, which is chosen ad hoc or by simple empirical tests. But it is actually a crucial variable, which poses an intricate trade-off that affects the optimizer's performance. On the one hand, the variance of the stochastic gradients *decreases linearly* with m, so small batches give vague gradient information, thus slow convergence in the number of optimization steps. On the other hand, the cost per step *increases linearly* with m. (This assumes that batch sizes are large enough to fully utilize the available parallel computing resources, which can easily be guaranteed by enforcing an appropriate minimal batch size.) While we can thus linearly trade off variance and cost, the gradient variance does *not* linearly affect the performance of the optimizer; its effect depends on the local structure of the objective and interacts with other parameter choices of the optimizer, notably the learning rate. In general, there should thus be an *optimal* batch size that balances these two aspects. Choosing such good batch sizes is an important aspect in the design of a numerical optimizer.

Below, we propose an algorithm that adapts the batch size based on the gradient variance observed by the optimizer at runtime. The exact variance over the entire data set (7) is prohibitively costly to compute, but it can be estimated by the sample variance computed on a mini-batch. As will be described below, we only require the diagonal elements of $\Sigma(w)$, corresponding to the variances of the individual components. These can be estimated by

$$S(w) = \frac{1}{m} \sum_{i \in \mathcal{B}} \nabla \ell_i(w)^{\cdot 2} - g(w)^{\cdot 2} \in \mathbb{R}^d, \qquad (10)$$

where $^{\cdot 2}$ signifies an element-wise square.

1.2 NOTATION

The following discussion addresses the choice of batch size for a single SGD step, assuming that we are currently at some arbitrary but fixed point w in parameter space. For notational convenience, we will thus drop w from the notation and write $F = F(w)$, $\nabla F = \nabla F(w)$, et cetera.

2 RELATED WORK

The dynamic adaption of batch sizes has already attracted attention in other recent works. Friedlander & Schmidt (2012) derive decreasing series of bounds on the gradient variance that provably yield fast convergence rates with a *constant* learning rate, showing that an increasing batch size can replace a decreasing learning rate. To realize these bounds in practice, they propose to increase the batch size by a pre-specified constant factor in each iteration, without adaptation to (an estimate of) the gradient variance.

The prior works closest to ours in spirit are by Byrd et al. (2012) and De et al. (2017), who propose to adapt the batch size based on variance estimates. Their criterion is based on the observation that $-g$ is a descent direction if

$$\|g - \nabla F\| \leq \theta \|g\|, \quad \text{with} \quad 0 \leq \theta < 1 \qquad (11)$$

(proof in Appendix A). While the left-hand side of (11) is of course unknown, one *can* compute its expected square

$$\begin{aligned} \mathbf{E}\left[\|g - \nabla F\|^2\right] &= \sum_{j=1}^{d} \mathbf{E}\left[(g_j - \nabla F_j)^2\right] \\ &= \sum_{j=1}^{d} \frac{\sigma_{jj}^2}{m} = \frac{\mathrm{tr}(\Sigma)}{m}. \end{aligned} \qquad (12)$$

Consequently, (11) holds *in expectation* if $\mathrm{tr}(\Sigma)/m \leq \theta^2 \|g\|^2$ or (with equality)

$$m = \frac{1}{\theta^2} \frac{\mathrm{tr}(\Sigma)}{\|g\|^2}. \qquad (13)$$

While this is a practical and intuitive method, the "descent direction" criterion is agnostic of the actual step being taken, which depends on the learning rate α in addition to the direction $-g$. Moreover, the method introduces an additional free parameter θ. In this work we strive to alleviate these issues, while the resulting batch size adaptation rule will stay close to (13) in form and spirit.

A somewhat related line of research aims to reduce the variance of stochastic gradients by incorporating gradient information from previous iterations into the current gradient estimate. Notable methods are SVRG (Johnson & Zhang, 2013) and SAGA (Defazio et al., 2014). Both are not mini-batch methods, since they update after gradient evaluations on individual training examples (which are then modified using stored gradient information). However, two recent papers (Harikandeh et al., 2015; Daneshmand et al., 2016) combine these variance-reduced methods with increasing sample sizes, i.e., the effective size of the training set is increased over time. In both, a sample size schedule has to be pre-specified and is not adapted at runtime.

We note that another recent line of work on non-uniform sampling of training samples with the goal of variance reduction (including, but not limited to, Needell et al., 2014; Zhao & Zhang, 2015; Schmidt et al., 2015; Csiba & Richtárik, 2016) is orthogonal to our work, since it is concerned with the composition of batches rather than their size.

More generally, our work fits into a recent effort to automate or simplify the tuning of parameters in stochastic optimization algorithms, most notably the learning rate (Schaul et al., 2013; Mahsereci & Hennig, 2015).

3 COUPLED ADAPTIVE BATCH SIZE

We will cast the problem of finding a "good" batch size as maximizing a lower bound on the expected *gain* per computational cost for an individual optimization step. While the resulting rule is similar in form to (13), it provides a new interpretation and introduces an explicit interaction with the learning rate. This criterion will subsequently be simplified, removing all unknown quantities and free parameters from the equation.

3.1 MAXIMIZING A BOUND ON THE EXPECTED GAIN

We define the *gain* of the SGD step from w to $w_+ = w - \alpha g$ as the drop in function value, $F - F_+$, where $F_+ = F(w_+)$. In order to quantify this gain, we will assume that F has Lipschitz-continuous gradients, i.e., there is a constant $L > 0$ such that

$$\|\nabla F(u) - \nabla F(v)\| \leq L\|u - v\| \quad \forall u, v \in \mathbb{R}^d. \quad (14)$$

This is a standard assumption in the analysis of stochastic optimization algorithms, setting a not overly restrictive bound on how fast the gradient can change when moving in parameter space. As a consequence, the change in F from $v \in \mathbb{R}^d$ to $u \in \mathbb{R}^d$ is bounded (see, e.g., Bottou et al. (2016), Eq. 4.3) by

$$F(u) \leq F(v) + \nabla F(v)^T(u - v) + \frac{L}{2}\|u - v\|^2. \quad (15)$$

Inserting $v = w$ and $u = w_+ = w - \alpha g$ and rearranging yields a lower bound $\mathcal{G}$ on the gain:

$$F - F_+ \geq \mathcal{G} := \alpha \nabla F^T g - \frac{L\alpha^2}{2}\|g\|^2. \quad (16)$$

To derive the expectation of $\mathcal{G}$, recall from Equation (9) that $\mathbf{E}[g] = \nabla F$ and

$$\begin{aligned} \mathbf{E}\left[\|g\|^2\right] &= \sum_{j=1}^{d} \mathbf{E}\left[g_j^2\right] = \sum_{j=1}^{d} \left(\nabla F_j^2 + \frac{\sigma_{jj}^2}{m}\right) \\ &= \|\nabla F\|^2 + \frac{\mathrm{tr}(\Sigma)}{m}, \end{aligned} \quad (17)$$

where we used that, for $X \sim \mathcal{N}(\mu, \sigma^2)$, the second moment is $\mathbf{E}[X^2] = \mu^2 + \sigma^2$. Thus,

$$\mathbf{E}[\mathcal{G}] = \left(\alpha - \frac{L\alpha^2}{2}\right)\|\nabla F\|^2 - \frac{L\alpha^2}{2m}\mathrm{tr}(\Sigma). \quad (18)$$

The first term in (18) is the gain in absence of noise, determined by α and ∇F. It is reduced by a term that depends on the gradient variance and drops with m. We see from (18) that, for an expected descent, $\mathbf{E}[\mathcal{G}] > 0$, we require

$$\alpha < \frac{2\|\nabla F\|^2}{L\left(\|\nabla F\|^2 + \mathrm{tr}(\Sigma)/m\right)}, \quad (19)$$

which exhibits a clear relationship between learning rate and batch size. Small batch sizes require a small learning rate, while larger batch sizes enable larger steps. We will exploit this relationship later on by explicitly coupling the two parameters. As a side note, for zero variance, we recover the well-known condition $\alpha < 2/L$ that guarantees convergence of gradient descent in the deterministic case.

Obviously, the larger m, the larger $\mathbf{E}[\mathcal{G}]$, so that the deterministic case is optimal if we ignore computational cost. Since that cost scales linearly with m, the optimal batch size is the one that maximizes expected gain per cost,

$$\max_m \frac{\mathbf{E}[\mathcal{G}]}{m}. \quad (20)$$

A recent workshop paper (Pirotta & Restelli, 2016) used a similar idea, although on a different quantity (a statistical lower bound on the linearized improvement). In our setting, maximal gain per cost is achieved by (derivation in Appendix A)

$$m = \frac{2L\alpha}{2 - L\alpha} \frac{\operatorname{tr}(\Sigma)}{\|\nabla F\|^2}. \tag{21}$$

3.2 THE CABS CRITERION

The result in (21) poses two practical problems. First, the Lipschitz constant L is an unknown property of the objective function. Even more importantly, it is difficult to reliably and robustly estimate the squared norm of the true gradient $\|\nabla F\|^2$ from a single batch. One might be tempted to replace it with $\|g\|^2$, recovering a criterion similar to (13), but this is not an unbiased estimator for the true gradient norm, as Equation (17) shows. Depending on the noise level and, intriguingly, the batch size m, the second term in (17) can introduce a significant bias.

In an effort to address these practical problems, we propose to replace Eq. (21) with the following simpler rule, which we term the *Coupled Adaptive Batch Size* (CABS):

$$m = \alpha \frac{\operatorname{tr}(\Sigma)}{F}. \tag{22}$$

A formal justification for this simplification will be given in §3.3, but first we want to highlight some intuitive benefits of this batch size adaptation scheme.

A major advantage of the CABS rule, emphasized in its name, is the direct *coupling* of learning rate and batch size. We have established that a large learning rate demands large batches while a smaller, more cautious learning rate can be used with smaller batches (Equation 19). The CABS rule explicitly reflects this known relationship. Using CABS can thus be seen as "tailoring" the noise level to the learning rate the user has chosen. We show experimentally, see §5, that this makes finding a well-performing learning rate easier.

Apart from that, theoretical considerations (Friedlander & Schmidt, 2012) and experimental evidence show that it is beneficial to have small batches in the beginning and larger ones later in the optimization process. Hence, one may want to think of the denominators of (22), (21) and (13) as a measure of "optimization progress". The function value F used in our CABS rule is, by definition, *the* measure for training progress. The norm of the true gradient $\|\nabla F\|^2$ conveys similar information (even though it might be misleading near non-optimal stationary points like saddle points or plateaus), but can not simply be estimated by $\|g\|^2$ as previously noted. We have also investigated unbiased estimators for $\|\nabla F\|^2$ by correcting the bias in $\|g\|^2$ using the variance estimate S, but these turned out to be too unreliable in experiments. Additionally, Equation (17) also shows that using $\|g\|^2$ in the denominator leads to a disadvantageous feedback: larger batches cause $\|g\|^2$ to become smaller in expectation which, in turn, leads to larger batches according to (13) (and the other way round).

Readers who are rightly worried about the change of "unit" or "type" when replacing the gradient norm in (21) with the function value in (22) may find it helpful to consider the units of measure for the quantities in (22). Let $[w]$ and $[F]$ denote the units of the parameters and the objective function, respectively. The gradient has unit $[F]/[w]$ and its variance $[F]^2/[w]^2$. It is a key insight that a well-chosen learning rate has to be driven by quantities with unit $[w]^2/[F]$ (see MacKay (2003) §34.4 for more discussion). If this was not the case, the gradient descent update $-\alpha_k \nabla F(w_k)$ would not be *covariant*, i.e., independent of the units of measure of w and F. It is also evident in Newton's method: in the one-dimensional case, a Newton update step is $-F''(w_k)^{-1}F'(w_k)$, corresponding to a "learning rate" that is given by the inverse second derivative, having unit $[F]/[w]^2$. Putting it all together, the right-hand side of (22) has unit

$$\frac{[w]^2}{[F]} \frac{[F]^2/[w]^2}{[F]} = [1]. \tag{23}$$

Hence, the chosen batch size is invariant under rescaling of the objective.

Lastly, CABS realizes a bound on the gradient variance that is decreasing with the distance to optimality, similar to that in Theorem 2.5 of Friedlander & Schmidt (2012), which they have shown to guarantee convergence of SGD with a constant, non-decreasing learning rate.

3.3 MATHEMATICAL MOTIVATION FOR CABS

For a more systematic motivation for the CABS rule, we will show that it is approximately equal to (21), and hence optimal in the sense of (20), if we assume that F locally has an approximately scalar Hessian, i.e.,

$$\nabla^2 F(w) \approx hI, \quad h > 0. \tag{24}$$

First, note that under this assumption, the Lipschitz constant L is exactly h and the optimal batch size according to (21) becomes

$$m = \frac{2h\alpha}{2 - h\alpha} \frac{\operatorname{tr}(\Sigma)}{\|\nabla F\|^2}. \tag{25}$$

Furthermore, the second-order Taylor expansion of F around w now reads

$$F(u) \approx F(w) + \nabla F(w)^T (u - w) + \frac{h}{2}\|u - w\|^2. \tag{26}$$

We minimize both sides with respect to u. The left-hand side takes on the optimal value F_*. For the right-hand side, we set the gradient with respect to u, $\nabla F(w) + h(u - w)$, to zero, which yields the minimizer $u = -\nabla F(w)/h + w$. Inserting this back into (26) and rearranging yields

$$\|\nabla F(w)\|^2 \approx 2h(F(w) - F_*). \tag{27}$$

That is, we can replace the squared gradient norm with a scaled distance to optimality. Doing so in (25) reads

$$m = \frac{\alpha}{2 - h\alpha} \frac{\mathrm{tr}(\Sigma)}{F - F_*}. \tag{28}$$

We eliminate h from this equation by realizing that, under the scalar Hessian assumption, a good learning rate is $\alpha = 1/h$. It corresponds both to the Newton step, as well as to the optimal constant learning rate $1/L$ for gradient descent, given that (24) holds. Hence, if we assume a well-chosen learning rate with $h\alpha \approx 1$, then Eq. (28) further simplifies to

$$m = \alpha \frac{\mathrm{tr}(\Sigma)}{F - F_*}. \tag{29}$$

Assuming a scalar Hessian is, of course, a substantial simplification. The result can partly be generalized to the less restrictive assumption of μ-strong convexity, under which we still have (see Appendix A)

$$\|\nabla F\|^2 \geq 2\mu(F - F_*), \tag{30}$$

If the problem is well-conditioned in that μ and L are not too far from each other, the above argument carries through as an upper bound on the batch size.

To finally arrive at the CABS rule, we drop F_*. This is based on the assumption of a non-negative loss, which holds for all standard loss functions like least-squares or cross-entropy. In this case, $F \geq F - F_*$, i.e., the function value F is a non-trivial upper bound on the distance to optimality. If the optimum is close to zero, F will be a good proxy for $F - F_*$. If not, which is not uncommon, the denominator of the CABS rule has a small positive offset compared to (29), but this will not fundamentally alter its implications, as long as we do not come too close to the optimum, which is usually the case for even modestly complex problems. The more general form (29) can be used in lieu of (22) if one has access to a tighter lower bound on F_*, e.g., due to prior experience from similar problems. In fact, when the objective function includes an additive regularization term, we suggest to use the unregularized loss as a proxy for $F - F_*$.

4 PRACTICAL IMPLEMENTATION

We outline a practical implementation of the CABS criterion. Obviously, neither F nor $\mathrm{tr}(\Sigma)$ are known exactly at each individual SGD step, but estimates of both quantities can be obtained from a mini-batch. This is straightforward for the objective F. For the variance, we use the estimate S explained in Equation (10). Since S only estimates the diagonal elements of the covariance matrix, it is $\mathrm{tr}(\Sigma) \approx \|S\|_1$. Considerations on how to practically compute S can be found in §4.2.

4.1 MECHANICAL DETAILS

We realize the CABS criterion in a predictive manner, meaning that we do *not* find the exact batch size that satisfies (22) in each single optimization step. To achieve such an exact enforcement of their criterion, Byrd et al. (2012) and De et al. (2017) increase the batch size by a small increment whenever the criterion is not satisfied, and only then perform the update. This incremental computation introduces an overhead and, when the increment is small, can lead to under-utilization of computing resources. Instead, we leverage the observation that gradient variance and function value change only slowly from one optimization step to the next, which allows us to use our current estimates of F and $\mathrm{tr}(\Sigma)$ to set the batch size used for the *next* optimization step. It also allows for a smoothing of both quantities over multiple optimization steps. The estimates can be fairly noisy, especially that of $\mathrm{tr}(\Sigma)$ at small batch sizes. We use exponential moving averages (see Algorithm 1) to obtain more robust estimates.

The resulting batch size is rounded to the nearest integer and clipped at minimal and maximal batch sizes. A minimal batch size avoids under-utilization of the computational resources with very small batches and provides additional stability of the algorithm in the small-batch regime. A maximal batch size is necessary due to hardware limitations: In contemporary deep learning, GPU memory limits the number of samples that can be processed at once. Our implementation has such a limit but it was never reached in our experiments. We note in passing that *algorithmic* batch size (the number of training samples used to compute a gradient estimate before updating the parameters) and *computational* batch size (the number of training samples that are processed simultaneously) are in principle independent—a future implementation could split an algorithmic batch into feasible computational batches when necessary, freeing the algorithm from hardware-specific constraints. Algorithm 1 provides pseudo-code.

Algorithm 1 SGD with Coupled Adaptive Batch Size

Require: Learning rate α, initial parameters w_0, number of steps K, batch size bounds $(m_{\text{min}}, m_{\text{max}})$, running average constant $\mu = 0.95$

$w \leftarrow w_0$, $m \leftarrow m_{\text{min}}$, $F_{\text{avg}} \leftarrow 0$, $\xi \leftarrow 0$
for $k = 1, \ldots, K$ **do**
  Draw a mini-batch $\mathcal{B}$ of size m
  $F, g, S \leftarrow$ EVALUATE(w, $\mathcal{B}$)
  $w \leftarrow w - \alpha g$
  $\xi \leftarrow \mu\xi + (1-\mu)\|S\|_1$
  $F_{\text{avg}} \leftarrow \mu F_{\text{avg}} + (1-\mu)F$
  $m \leftarrow$ROUND_&_CLIP($\alpha\xi / F_{\text{avg}}$, m_{min}, m_{max})
end for

Note: EVALUATE(w, $\mathcal{B}$) denotes an evaluation of function value $F(w)$, stochastic gradient $g(w)$ and variance estimate $S(w)$ (Eq. 10) using mini-batch $\mathcal{B}$. ROUND_&_CLIP(m, m_{min}, m_{max}) rounds m to the nearest integer and clips it at the provided minimal and maximal values.

4.2 VARIANCE ESTIMATE

If the individual gradients $\nabla\ell_i$ in the mini-batch are accessible, then S can be computed directly by Eq. (10), adding only the computational cost of squaring and summing the gradients. Unfortunately, these individual gradients are not available in practical implementations of the backpropagation algorithm (Rumelhart et al., 1986) used to compute gradients in the training of neural networks. A complete discussion of this technical issue is beyond the scope of this paper, but we briefly sketch a solution.

Consider a fully-connected layer in a neural network with weight matrix $W_{l+1} \in \mathbb{R}^{n_l \times n_{l+1}}$. During the forward pass, the matrix of activations $A_l \in \mathbb{R}^{m \times n_l}$ (containing the activations for each of the m input training samples) is propagated forward by a matrix multiplication,

$$Z_{l+1} = A_l W_{l+1} \in \mathbb{R}^{m \times n_{l+1}}. \tag{31}$$

Once the backward pass arrives at this layer, the gradient with respect to W_{l+1} is computed as

$$dW_{l+1} = A_l^T \, dZ_{l+1}. \tag{32}$$

The aggregation of individual gradients is *implicit* in this matrix multiplication. Practical implementations rely on the efficiency of these matrix operations and, even more importantly, it is infeasible to store m individual gradients in memory if the number of parameters d is high.

However, one can similarly compute the *second moment* of the gradients, $\frac{1}{m}\sum_i (\nabla\ell_i)^{\cdot 2}$, that is needed in (10) without giving up efficient batch processing. It is straight-forward to verify that this second moment of gradients with respect to $W^{(l+1)}$ can be computed as

$$\left(A_l^T\right)^{\cdot 2} \left(dZ_{l+1}\right)^{\cdot 2}. \tag{33}$$

In this form, the computation of the gradient variance adds non-negligible but manageable computational cost. Since it duplicates half of the operations in the backward pass, the additional cost can be pinned down to roughly 25%. This is primarily an implementation issue; the cost could be reduced by implementing special matrix operations to compute (32) and (33) jointly.

5 EXPERIMENTS

We evaluate the proposed batch size adaptation method by training convolutional neural networks (CNNs) on four popular image classification benchmark data sets: MNIST (LeCun et al., 1998), Street View House Numbers (SVHN) (Netzer et al., 2011), as well as CIFAR-10 and CIFAR-100 (Krizhevsky, 2009). While these are small to medium-scale problems by contemporary standards, they exhibit many of the typical difficulties of neural network training. We opted for these benchmarks to keep the computational cost for a *thorough* evaluation of the method manageable (this required approximately 60 training runs per benchmark, see the following section).

5.1 EXPERIMENT DESIGN

We compare against constant batch sizes 16, 32, 64, 128, 256 and 512. To keep the plots readable, we only report results for batch sizes 32, 128 and 512 in the main text; results for the other batch sizes can be found in the supplements. We also compare against a batch size adaptation based on the criterion (13) used in Byrd et al. (2012) and De et al. (2017). Since implementation details differ between these two works, and both combine batch size adaptation with other measures (Newton-CG method in Byrd et al. (2012) and a backtracking line search in De et al. (2017)), we resort to a custom implementation of said criterion. For a fair comparison, we realize it in a similar manner as CABS. That is, we use criterion (13), while keeping the predictive update mechanism for the batch size, the smoothing via exponential moving averages, rounding and clipping exactly as in our CABS implementation described in §4.1 and Algorithm 1. This method will simply be referred to as *Competitor* in the remainder of this section.

During the optimization process, we periodically evaluate the training loss as well as the classification accuracy on a held-out test set. Since each method uses a different batch size, both quantities are tracked as a function of the number of accessed training examples, instead of the

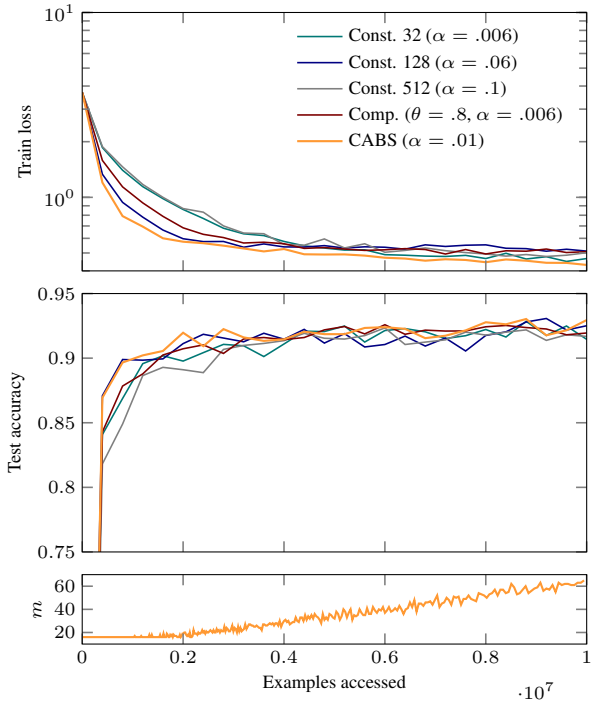

Figure 1: Results for SVHN. Shared horizontal axis indicates the number of examples used for training. Top and middle panel depict evolution of training loss and test accuracy, respectively, color-coded for different batch size methods, each with its optimal learning rate. Bottom panel shows batch size chosen by CABS.

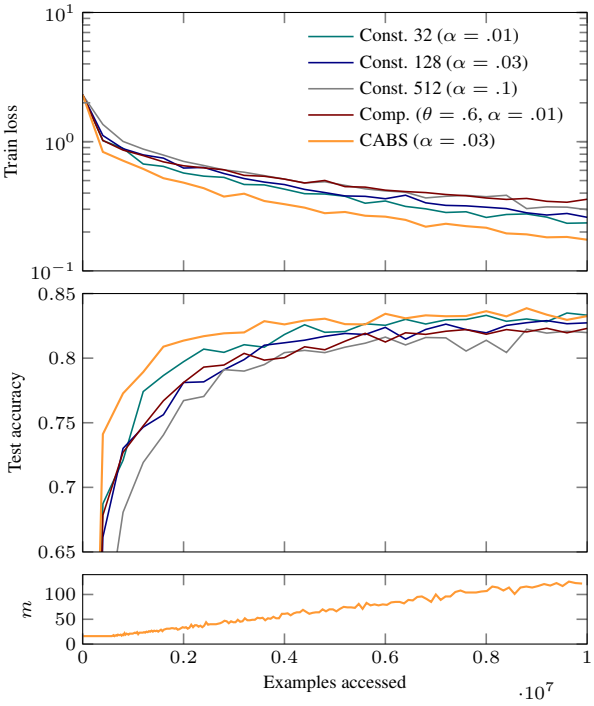

Figure 2: Results for CIFAR-10. Set-up as in Fig. 1.

number of optimization steps. This measure is proportional to wall-clock time up to per-batch overheads that depend on the specific problem and implementation.

The (constant) learning rate for each batch size method was tuned for maximum test accuracy given the fixed budget of accessed training examples. We tried six candidates $\alpha \in \{0.3, 0.1, 0.06, 0.03, 0.01, 0.006\}$; this relevant range has been determined with a few exploratory experiments. In addition to the learning rate, the competitor method has a free parameter θ. De et al. (2017) suggest setting it to 1.0, the highest possible noise tolerance, by default. In our experiments, we found the performance of the method to be fairly sensitive to the choice of θ. We thus tried $\theta \in \{0.6, 0.8, 1.0\}$ and report results for the best-performing choice. For CABS, there is no analogous parameter to tune.

MNIST We start with experiments on the well-known MNIST image classification task of identifying handwritten digits in 28×28 pixel gray scale images. Our network has two convolutional layers with 5×5 filters (32 and 64 filters, respectively) and subsequent max-pooling over 2×2 windows. This is followed by a fully-connected layer with 1024 units. The activation function is ReLU for all layers. The output layer has 10 units with softmax activation and we use cross-entropy loss.

SVHN Next, we train a CNN on the digit classification task of the Street View House Numbers (SVHN) data set. While the task is similar to MNIST, the images are in RGB and larger (32×32). They exhibit real-world views of digits in house numbers, partially with clutter, misalignment and distracting digits at the sides. We train a CNN with two convolutional layers, each with 64 filters of size 5×5 and subsequent max-pooling over 3×3 windows with stride 2. They are followed by two fully-connected layers with 256 and 128 units, respectively. The activation function is ReLU for all layers. The output layer has 10 units with softmax activation and we use cross-entropy loss. We apply L_2-regularization and perform data augmentation operations (random cropping of 24×24 pixel subimages, left-right mirroring, color distortion) on the training inputs.

CIFAR-10 and CIFAR-100 Finally, we train CNNs on the CIFAR-10 and CIFAR-100 data sets, where the task is to classify 32×32 pixel RGB images into one of 10 and 100 object categories, respectively. For CIFAR-10, we crop the images to 24×24 pixels and train a CNN with two convolutional layers, each with 64 filters of size

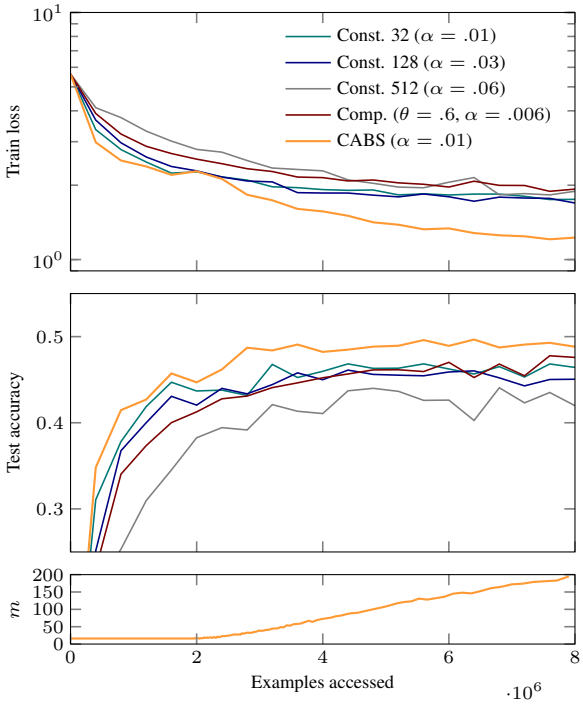

Figure 3: Results for CIFAR-100. Set-up as in Fig. 1.

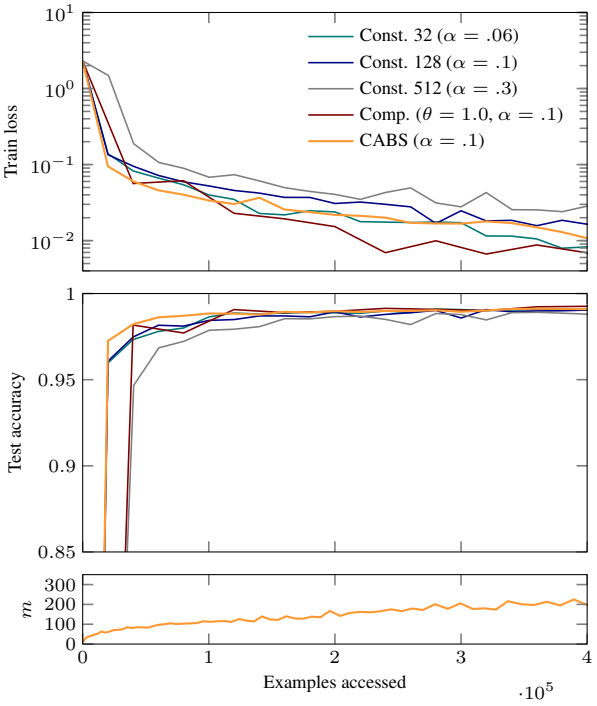

Figure 4: Results for MNIST. Set-up as in Fig. 1.

5x5 and subsequent max-pooling over 3x3 windows with stride 2. They are followed by two fully-connected layers with 384 and 192 units, respectively. The activation function is ReLU for all layers. The output layer has 10 units with softmax activation and we use cross-entropy loss. We perform data augmentation operations (random cropping, left-right mirroring, color distortion) on the training set.

For CIFAR-100, we use the full 32×32 image and add a third convolutional layer (64 filters of size 5×5 followed by max pooling). The fully-connected layers have 512 and 256 units, respectively, and the output layer has 100 units. We add L_2-regularization.

5.2 RESULTS AND DISCUSSION

On SVHN, CIFAR-10 and CIFAR-100 (Figures 1, 2 and 3), CABS yields significantly faster decrease in training loss with the curve contiuously lying below all others. It also achieves the best test set accuracy of all methods on all three problems. While the margin over the second-best method is very small on SVHN, it amounts to a noticeable 0.4 percentage points on CIFAR-10 and even 1.4 points on CIFAR-100.

Surprisingly, on MNIST—the *least* complex of the benchmark problems we investigated—our method is outperformed by the small constant batch size of 32 and the competitor method, which also chooses very small batch sizes throughout. Our method does, however, surpass non-adaptive larger batch sizes (128, 512) in terms of speed and all contenders reach virtually the same test set accuracy on this problem. CABS makes rapid progress initially, but seems to choose unnecessarily large batch sizes later on. The resulting high per-iteration cost evidently can not be compensated by the higher learning rate it enables (.1 for CABS compared to .01 with constant batch size of 32). We conjecture that CABS overestimates the gradient variance due to the homogeneous structure of the MNIST data set; if the distribution of gradients is very closely-centered, outliers in a few coordinate directions lead to comparably high variance estimates.

Overall, CABS outperforms alternative batch size schemes on three out of the four benchmark problems we investigated and the benefits seem to increase with the complexity of the problem (MNIST $\rightarrow$ SVHN $\rightarrow$ CIFAR-10 $\rightarrow$ CIFAR-100). When considering the CABS batch size schedules, depicted in the bottom panels of Figures 1 to 4, a common behavior (with the exception of MNIST) seems to be that CABS uses the minimal batch size (16 in our experiments) for a considerable portion of the training process and increases approximately linearly afterwards.

Finally, we present our findings regarding the sensitivity to the choice of learning rate when using CABS. As

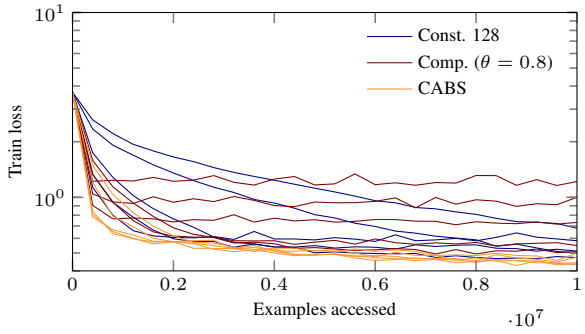

Figure 5: Learning rate sensitivity on SVHN. Families of training loss curves for CABS, Competitor and a constant batch size (color-coded). Each individual curve corresponds to a learning rate $\alpha \in \{.1, .06, .03, .01, .006\}$.

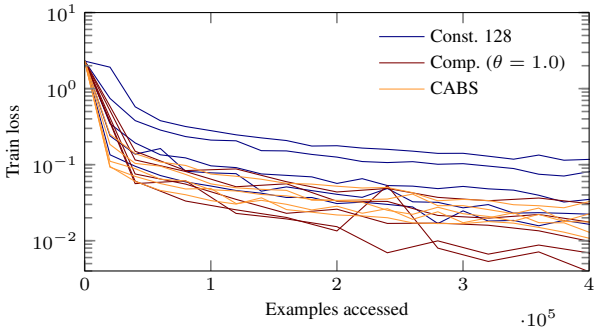

Figure 6: Learning rate sensitivity on MNIST. Set-up as in Fig. 5.

detailed above, the coupling of learning rate and batch size in CABS can be seen as tayloring the noise level to the chosen learning rate. This suggests that the performance of the optimizer should be less sensitive to the choice of learning rate when adapting the batch size with our method. Indeed, this became evident in our experiments by considering the families of training loss curves for various learning rates. Figures 5 and 6 compare a constant batch size, the competitor method and CABS on the SVHN and MNIST benchmark problems, respectively. CABS significantly reduces the dependency of the performance on the learning rate compared to both the constant batch size and the competing adaptive method. In practical applications, this finding could drastically simplify the often tedious process of learning rate tuning.

6 CONCLUSION

We proposed CABS, a practical rule for dynamic batch size adaptation based on estimates of the gradient variance and coupled to the chosen learning rate as well as optimization progress represented by the function value. In our experiments, CABS was able to speed up SGD training in neural networks and simplify the tuning of the learning rate. In contrast to existing methods, it does not introduce any additional free parameters. A TensorFlow[1] implementation of SGD with CABS can be found on `http://github.com/ProbabilisticNumerics/cabs`.

A MATHEMATICAL DETAILS

Proof of Equation (11) By the Cauchy-Schwarz inequality, $\langle g, \nabla F\rangle = \langle g, g\rangle - \langle g, g - \nabla F\rangle \geq \|g\|^2 - \|g\|\|g - \nabla F\| = \|g\|(\|g\| - \|g - \nabla F\|)$. This becomes positive if $\|g - \nabla F\| < \|g\|$.

Solving the Maximization Problem (20) We want to maximize

$$U(m) = \frac{\mathbf{E}[\mathcal{G}]}{m} = \frac{2\alpha - L\alpha^2}{2m}\|\nabla F\|^2 - \frac{L\alpha^2}{2m^2}\mathrm{tr}(\Sigma). \tag{34}$$

Setting the derivative

$$U'(m) = -\frac{2\alpha - L\alpha^2}{2m^2}\|\nabla F\|^2 + \frac{L\alpha^2}{m^3}\mathrm{tr}(\Sigma) \tag{35}$$

to zero and rearranging yields (21).

Proof of Equation (30) The definition of strong convexity is that, for all $w, u \in \mathbb{R}^d$

$$F(u) \geq F(w) + \nabla F(w)^T(u - w) + \frac{\mu}{2}\|u - w\|^2 \tag{36}$$

for $\mu > 0$. From there, the proof is identical to that of Equation (27) in the main text. We minimize both sides of the inequality. The left-hand side has minimal value F_*. The gradient with respect to u of the right-hand side is $\nabla F(w) + \mu(u - w)$. By setting this to zero we find the minimizer $u = -\nabla F(w)/\mu + w$. Inserting this back into (36) and rearranging yields (30). □

References

Bottou, Léon, Curtis, Frank E, and Nocedal, Jorge. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Byrd, Richard H, Chin, Gillian M, Nocedal, Jorge, and Wu, Yuchen. Sample size selection in optimization methods for machine learning. *Mathematical Programming*, 134(1):127–155, 2012.

Csiba, Dominik and Richtárik, Peter. Importance sampling for minibatches. *arXiv preprint arXiv:1602.02283*, 2016.

[1] `http://tensorflow.org`

Daneshmand, Hadi, Lucchi, Aurelien, and Hofmann, Thomas. Starting small—learning with adaptive sample sizes. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, pp. 1463–1471, 2016.

De, Soham, Yadav, Abhay, Jacobs, David, and Goldstein, Tom. Automated inference with adaptive batches. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Defazio, Aaron, Bach, Francis, and Lacoste-Julien, Simon. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*, pp. 1646–1654. 2014.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Friedlander, Michael P and Schmidt, Mark. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

Harikandeh, Reza, Ahmed, Mohamed Osama, Virani, Alim, Schmidt, Mark, Konečný, Jakub, and Sallinen, Scott. StopWasting my gradients: Practical SVRG. In *Advances in Neural Information Processing Systems 28*, pp. 2251–2259. 2015.

Johnson, Rie and Zhang, Tong. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pp. 315–323. 2013.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *The International Conference on Learning Representations (ICLR)*, 2015.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

MacKay, David JC. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

Mahsereci, Maren and Hennig, Philipp. Probabilistic line searches for stochastic optimization. In *Advances in Neural Information Processing Systems 28*, pp. 181–189. 2015.

Needell, Deanna, Ward, Rachel, and Srebro, Nati. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems 27*, pp. 1017–1025. 2014.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Pirotta, Matteo and Restelli, Marcello. Cost-sensitive approach for batch size optimization. *NIPS Workshop on Optimizing the Optimizer*, 2016.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Schaul, Tom, Zhang, Sixin, and LeCun, Yann. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 343–351, 2013.

Schmidt, Mark, Babanezhad, Reza, Ahmed, Mohamed Osama, Defazio, Aaron, Clifton, Ann, and Sarkar, Anoop. Non-uniform stochastic average gradient method for training conditional random fields. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

Zeiler, Matthew D. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhao, Peilin and Zhang, Tong. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of The 32nd International Conference on Machine Learning (ICML)*, 2015.