The Power of Interpolation: Understanding the Effectiveness of SGD in Modern Over-parametrized Learning[†]

Siyuan Ma [1] **Raef Bassily** [1] **Mikhail Belkin** [1]

Abstract

In this paper we aim to formally explain the phenomenon of fast convergence of Stochastic Gradient Descent (SGD) observed in modern machine learning. The key observation is that most modern learning architectures are over-parametrized and are trained to interpolate the data by driving the empirical loss (classification and regression) close to zero. While it is still unclear why these interpolated solutions perform well on test data, we show that these regimes allow for fast convergence of SGD, comparable in number of iterations to full gradient descent. For convex loss functions we obtain an exponential convergence bound for *mini-batch* SGD parallel to that for full gradient descent. We show that there is a critical batch size m^* such that: (a) SGD iteration with mini-batch size $m \leq m^*$ is nearly equivalent to m iterations of mini-batch size 1 (*linear scaling regime*). (b) SGD iteration with mini-batch $m > m^*$ is nearly equivalent to a full gradient descent iteration (*saturation regime*). Moreover, for the quadratic loss, we derive explicit expressions for the optimal mini-batch and step size and explicitly characterize the two regimes above. The critical mini-batch size can be viewed as the limit for effective mini-batch parallelization. It is also nearly independent of the data size, implying $O(n)$ acceleration over GD per unit of computation. We give experimental evidence on real data which closely follows our theoretical analyses. Finally, we show how our results fit in the recent developments in training deep neural networks and discuss connections to adaptive rates for SGD and variance reduction.

1 Introduction

Most machine learning techniques for supervised learning are based on Empirical Loss Minimization (ERM), i.e., minimizing the loss $\mathscr{L}(\boldsymbol{w}) \triangleq \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{w})$ over some parametrized space of functions $f_{\boldsymbol{w}}$. Here $\ell_i(\boldsymbol{w}) = L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i)$, where $(\boldsymbol{x}_i, y_i)$ are the data and L could, for example, be the square loss $L(f_{\boldsymbol{w}}(\boldsymbol{x}), y) = (f_{\boldsymbol{w}}(\boldsymbol{x}) - y)^2$.

In recent years, Stochastic Gradient Descent (SGD) with a small mini-batch size has become the backbone of machine learning, used in nearly all large-scale applications of machine learning methods, notably in conjunction with deep neural networks. Mini-batch SGD is a first order method which, instead of computing the full gradient of $\mathscr{L}(\boldsymbol{w})$, computes the gradient with respect to a certain subset of the data points, often chosen sequentially. In practice small mini-batch SGD consistently outperforms full gradient descent (GD) by a large factor in terms of the computations required to achieve certain accuracy. However, the theoretical evidence has been mixed. While SGD needs less computations per iteration, most analyses suggest that it requires adaptive step sizes and has the rate of convergence that is far slower than that of GD, making computational efficiency comparisons difficult.

In this paper, we explain the reasons for the effectiveness of SGD by taking a different perspective. We note that most of modern machine learning, especially deep learning, relies on classifiers which are trained to achieve near zero classification and regression losses on the training data. Indeed, the goal of achieving near-perfect fit on the training set is stated explicitly by the practitioners as a best practice in supervised learning[1], see, e.g., the tutorial (Salakhutdinov, 2017). The ability to achieve near-zero loss is provided by over-parametrization. The number of parameters for most deep architectures is very large and often exceeds by far the size of the datasets used for training (see, e.g., (Canziani et al., 2016) for a summary of different architectures). There is significant theoretical and empirical evidence that in such over-parametrized systems most or all local minima are also global and hence correspond to the regime where the output of the learning algorithm matches the training labels exactly,

[1] Potentially using regularization at a later stage.

† See full version of this paper at arxiv.org/abs/1712.06559.

[1]Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA. Correspondence to: Siyuan Ma <masi@cse.ohio-state.edu>, Raef Bassily <bassily.1@osu.edu>, Mikhail Belkin <mbelkin@cse.ohio-state.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

e.g., (Gupta et al., 2015; Chaudhari et al., 2016; Zhang et al., 2016; Huang et al., 2016; Sagun et al., 2017; Bartlett et al., 2017). Since continuous loss functions are typically used for training, the resulting function *interpolates* the data[2], i.e., $f_{\boldsymbol{w}^*}(\boldsymbol{x}_i) \approx y_i$.

While we do not yet understand why these interpolated classifiers generalize so well to unseen data, there is ample empirical evidence for their excellent generalization performance in deep neural networks (Gupta et al., 2015; Chaudhari et al., 2016; Zhang et al., 2016; Huang et al., 2016; Sagun et al., 2017), kernel machines (Belkin et al., 2018) and boosting (Schapire et al., 1998). In this paper we look at the significant computational implications of this startling phenomenon for stochastic gradient descent.

Our first key observation is that in the interpolated regime SGD with fixed step size converges exponentially fast for convex loss functions. The results showing exponential convergence of SGD when the optimal solution minimizes the loss function at each point go back to the Kaczmarz method (Kaczmarz, 1937) for quadratic functions, more recently analyzed in (Strohmer & Vershynin, 2009). For the general convex case, it was first proved in (Moulines & Bach, 2011). The rate was later improved in (Needell et al., 2014). However, to the best of our knowledge, exponential convergence in that regime has not been connected to over-parametrization and interpolation in modern machine learning. Still, exponential convergence by itself does not allow us to make any comparisons between the computational efficiency of SGD with different mini-batch sizes and full gradient descent, as the existing results do not depend on the mini-batch size m. This dependence is crucial for understanding SGD, as small mini-batch SGD seems to dramatically outperform full gradient descent in nearly all applications. Motivated by this, in this paper we provide an explanation for the empirically observed efficiency of small mini-batch SGD. We provide a detailed analysis for the rates of convergence and computational efficiency for different mini-batch sizes and a discussion of its implications in the context of modern machine learning.

We first analyze convergence of mini-batch SGD for convex loss functions as a function of the batch size m. We show that there is a critical mini-batch size m^* that is nearly independent on n, such that the following holds:
(a) (linear scaling) One SGD iteration with mini-batch of size $m \leq m^*$ is equivalent to m iterations of mini-batch of size one up to a multiplicative constant close to 1.
(b) (saturation) One SGD iterations with a mini-batch of size $m > m^*$ is nearly (up to a small constant) as effective as one iteration of full gradient descent.

[2]Most of these architectures should be able to achieve perfect interpolation, $f_{\boldsymbol{w}^*}(\boldsymbol{x}_i) = y_i$. In practice, of course, it is not possible even for linear systems due to the computational and numerical limitations.

We see that the critical mini-batch size m^* can be viewed as the limit for the effective parallelization of mini-batch computations. If an iteration with mini-batch of size $m \leq m^*$ can be computed in parallel, it is nearly equivalent to m sequential steps with mini-batch of size 1. For $m > m^*$ parallel computation has limited added value.

Next, for the quadratic loss function, we obtain a sharp characterization of these regimes based on an explicit derivation of optimal step size as a function of m. In particular, in this case we show that the critical mini-batch size is given by $m^* = \frac{\max_{i=1}^n \{\|\boldsymbol{x}_i\|^2\}}{\lambda_1(H)}$, where H is the Hessian at the minimizer and λ_1 is its spectral norm.

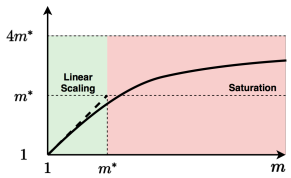


Figure 1: Number of iterations with batch size 1 (the y axis) equivalent to one iteration with batch size m.

Our result shows that m^* is nearly independent of the data size n (depending only on the properties of the Hessian). Thus SGD with mini-batch size m^* (typically a small constant) gives essentially the same convergence per iteration as full gradient descent, implying acceleration by a factor of $O(n)$ over GD per unit of computation.

We also show that a mini-batch of size one is optimal in terms of computations required to achieve a given error. Our theoretical results are based on upper bounds which we show to be tight in the quadratic case and nearly tight in the general convex case.

There have been work on understanding the interplay between the mini-batch size and computational efficiency, including (Takác et al., 2013; Li et al., 2014; Yin et al., 2018) in the standard non-interpolated regime. However, in that setting the issue of bridging the exponential convergence of full GD and the much slower convergence rates of mini-batch SGD is harder to resolve, requiring extra components, such as tail averaging (Jain et al., 2016) (for quadratic loss).

We provide experimental evidence corroborating this on real data. In particular, we demonstrate the regimes of linear scaling and saturation and also show that on real data m^* is in line with our estimate. It is typically several orders of magnitude smaller than the data size n implying a computational advantage of at least 10^3 factor over full gradient descent in realistic scenarios in the over-parametrized (or fully parametrized) setting. We believe this sheds light on the impressive effectiveness of SGD observed in many real-world situation and is the reason why full gradient descent is rarely, if ever, used. In particular, the "linear scaling rule" recently used in deep convolutional networks (Krizhevsky, 2014; Goyal et al., 2017; You et al., 2017; Smith et al., 2017) is consistent with our theoretical analyses.

The rest of the paper is organized as follows: In Section 3, we analyze the fast convergence of mini-batch SGD and discuss some implications for the variance reduction techniques. It turns out that in the interpolated regime, simple SGD with constant step size is equally or more effective than the more complex variance reduction methods. Section 4 contains the analysis of the special case of quadratic losses, where we obtain optimal convergence rates of mini-batch SGD, and derive the optimal step size as a function of the mini-batch size. We also analyze the computational efficiency as a function of the mini-batch size. In Section 5 we provide experimental evidence using several datasets. We show that the experimental results correspond closely to the behavior predicted by our bounds. We also briefly discuss the connection to the linear scaling rule in neural networks.

2 Preliminaries

Before we start our technical discussion, we briefly overview some standard notions in convex analysis. Here, we will focus on differentiable convex functions, however, the definitions below extend to general functions simply by replacing the gradient of the function at a given point to by the set of all sub-gradients at that point. In fact, since in this paper we only consider smooth functions, differentiability is directly implied.

- A differentiable function $\ell : \mathbb{R}^d \to \mathbb{R}$ is *convex* on $\mathbb{R}^d$ if, for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$, we have $\ell(\boldsymbol{v}) \geq \ell(\boldsymbol{w}) + \langle \nabla \ell(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle$.
- Let $\beta > 0$. A differentiable function $\ell : \mathbb{R}^d \to \mathbb{R}$ is *β-smooth* on $\mathbb{R}^d$ if, for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$, we have $\ell(\boldsymbol{v}) \leq \ell(\boldsymbol{w}) + \langle \nabla \ell(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle + \frac{\beta}{2} \|\boldsymbol{v} - \boldsymbol{w}\|^2$, where $\nabla \ell(\boldsymbol{w})$ denotes the gradient of ℓ at $\boldsymbol{w}$.
- Let $\alpha > 0$. A differentiable function $\ell : \mathbb{R}^d \to \mathbb{R}$ is *α-strongly convex* on $\mathbb{R}^d$ if, for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$, we have $\ell(\boldsymbol{v}) \geq \ell(\boldsymbol{w}) + \langle \nabla \ell(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle + \frac{\alpha}{2} \|\boldsymbol{v} - \boldsymbol{w}\|^2$. (Clearly, α-strong convexity implies convexity)

The problem of unconstrained Empirical Risk Minimization (ERM) can be described as follows: Given a set of n loss functions $\ell_i : \mathbb{R}^d \to \mathbb{R}$, $i \in \{1, \ldots, n\}$, the goal is to minimize the empirical loss function defined as

$$\mathscr{L}(\boldsymbol{w}) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell_i(\boldsymbol{w}), \ \boldsymbol{w} \in \mathbb{R}^d.$$

In particular, we want to find a minimizer $\boldsymbol{w}^* \triangleq \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \mathscr{L}(\boldsymbol{w})$. In the context of supervised learning, given a training set $\{(\boldsymbol{x}_i, y_i) : 1 \leq i \leq n\}$ of n (feature vector, target) pairs, one can think of $\ell_i(\boldsymbol{w})$ as the cost incurred in choosing a parameter vector $\boldsymbol{w}$ to fit the data point $(\boldsymbol{x}_i, y_i)$. In particular, in this context, minimizing $\mathscr{L}$ over $\boldsymbol{w} \in \mathbb{R}^d$ is equivalent to minimizing $\mathscr{L}$ over a parameterized space of functions $\{f_{\boldsymbol{w}} : \boldsymbol{w} \in \mathbb{R}^d\}$, where each $f_{\boldsymbol{w}}$ maps a feature vector $\boldsymbol{x}$ to a target y. Thus, in this case, for each i, $\ell_i(\boldsymbol{w})$ can be written as $L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i)$ where L is some cost function that represents how far is $f_{\boldsymbol{w}}(\boldsymbol{x}_i)$ from y_i, for example, $L(\cdot, \cdot)$ could be the squared loss $L(f_{\boldsymbol{w}}(\boldsymbol{x}), y) = (f_{\boldsymbol{w}}(\boldsymbol{x}) - y)^2$.

3 Interpolation and Fast SGD: Convex Loss

We consider a standard setting of ERM where for all $1 \leq i \leq n$, ℓ_i is non-negative, β-smooth and convex. Moreover, $\mathscr{L}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(\boldsymbol{w})$ is λ-smooth and α-strongly convex. It is easy to see that $\beta \geq \lambda$. This setting is naturally satisfied in many problems, e.g., in least-squares linear regression with full rank sample covariance matrix.

Next, we state our key assumption in this work. This assumption describes the interpolation setting, which is aligned with what we usually observe in over-parametrized settings in modern machine learning.

Assumption 1 (Interpolation)**.** Let $\boldsymbol{w}^* \in \operatorname{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \mathscr{L}(\boldsymbol{w})$. Then, for all $1 \leq i \leq n$, $\ell_i(\boldsymbol{w}^*) = 0$.

Note that instead of assuming that $\ell_i(\boldsymbol{w}^*) = 0$, it suffices to assume that $\boldsymbol{w}^*$ is the minimizer of all ℓ_i. By subtracting from each ℓ_i the offset $\ell_i(\boldsymbol{w}^*)$, we get an equivalent minimization problem where the new losses are all non-negative, and are all zero at $\boldsymbol{w}^*$.

Consider the SGD algorithm that starts at an arbitrary $\boldsymbol{w}_0 \in \mathbb{R}^d$, and at each iteration t makes an update with a constant step size η:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla \frac{1}{m} \sum_{j=1}^{m} \ell_{i_t^{(j)}}(\boldsymbol{w}_t) \tag{1}$$

where m is the size of a mini-batch of data points whose indices $\{i_t^{(1)}, \ldots, i_t^{(m)}\}$ are drawn uniformly with replacement at each iteration t from $\{1, \ldots, n\}$.

The theorem below shows exponential convergence for mini-batch SGD in the interpolated regime.

Theorem 1. *For the setting described above and under Assumption 1, for any mini-batch size $m \in \mathbb{N}$, the SGD iteration (1) with constant step size $\eta^*(m) \triangleq \frac{m}{\beta + \lambda(m-1)}$ gives the following guarantee*

$$\mathop{\mathbb{E}}_{\boldsymbol{w}_t} [\mathscr{L}(\boldsymbol{w}_t)] \leq \frac{\lambda}{2} (1 - \eta^*(m) \cdot \alpha)^t \|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 \tag{2}$$

For $m = 1$, this theorem is a special case of Theorem 2.1 in (Needell et al., 2014), which is a sharper version of Theorem 1 in (Moulines & Bach, 2011).

Speedup factor. Let $t(m)$ be the number of iterations needed to reach a desired accuracy with batch size m. Assuming $\lambda \gg \alpha$, the speed up factor $\frac{t(1)}{t(m)}$, which measures

the number of iterations saved by using larger batch, is

$$\frac{t(1)}{t(m)} = \frac{\log(1-\eta^*(m)\alpha)}{\log(1-\eta^*(1)\alpha)} \approx \frac{\eta^*(m)}{\eta^*(1)} = \frac{m\beta}{\beta+\lambda(m-1)}$$

Critical batch size $m^* \triangleq \frac{\beta}{\lambda}+1$. By estimating the speedup factor for each batch size m, we directly obtain

- Linear scaling regime: one iteration of batch size $m \leq m^*$ is nearly equivalent to m iterations of batch size 1.
- Saturation regime: one iteration with batch size $m > m^*$ is nearly equivalent to one full gradient iteration.

We give a sharper analysis for the case of quadratic loss in Section 4.

3.1 Variance reduction methods in the interpolation regime

For general convex optimization, a set of important stochastic methods (Roux et al., 2012; Johnson & Zhang, 2013; Defazio et al., 2014; Xiao & Zhang, 2014; Allen-Zhu, 2016) have been proposed to achieve exponential (linear) convergence rate with constant step size. The effectiveness of these methods derives from their ability to reduce the stochastic variance caused by sampling. In a general convex setting, this variance prevents SGD from both adopting a constant step size and achieving an exponential convergence rate.

Method	Step size	#Iterations to reach a given error
Mini-batch SGD (Theorem 1)	$\frac{m}{\beta+\lambda(m-1)}$	$O(\frac{\beta+\lambda(m-1)}{m\alpha})$
SGD (Eq. 5, m=1)	$\frac{1}{\beta}$	$O(\frac{\beta}{\alpha})$
SAG (Roux et al., 2012)	$\frac{1}{2n\cdot\beta}$	$O(\frac{n\cdot\beta}{\alpha})$
SVRG (Johnson & Zhang, 2013)	$\frac{1}{10\beta}$	$O(n+\frac{\beta}{\alpha})$
SAGA (Defazio et al., 2014)	$\frac{1}{3\beta}$	$O(n+\frac{\beta}{\alpha})$
Katyusha (Allen-Zhu, 2016) (momentum)	adaptive	$O(n+\sqrt{\frac{n\cdot\beta}{\alpha}})$

Remarkably, in the interpolated regime, Theorem 1 implies that SGD obtains the benefits of variance reduction "for free" without the need for any modification or extra information (e.g., full gradient computations for variance reduction). The table on the right compares the convergence of SGD in the interpolation setting with several popular variance reduction methods. Overall, SGD has the largest step size and achieves the fastest convergence rate without the need for any further assumptions. The only comparable or faster rate is given by Katyusha, which is an accelerated SGD method combining momentum and variance reduction for faster convergence.

4 How Fast is Fast SGD: Analysis of Step, Mini-batch Sizes and Computational Efficiency for Quadratic Loss

In this section, we analyze the convergence of mini-batch SGD for quadratic losses. We will consider the following key questions:

- What is the optimal convergence rate of mini-batch SGD and the corresponding step size as a function of m (size of mini-batch)?
- What is the computational efficiency of different batch sizes and how do they compare to full GD?

The case of quadratic losses covers over-parametrized linear or kernel regression with a positive definite kernel. The quadratic case also captures general smooth convex functions in the neighborhood of a minimum where higher order terms can be ignored.

Quadratic loss. Consider the problem of minimizing,

$$\mathscr{L}(\boldsymbol{w}) \triangleq \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{w}^T\boldsymbol{x}_i - y_i)^2$$

where $(\boldsymbol{x}_i, y_i) \in \mathscr{H} \times \mathbb{R}, i = 1,\ldots,n$ are labeled data points sampled from some (unknown) distribution. In the interpolation setting, there exists $\boldsymbol{w}^* \in \mathscr{H}$ such that $\mathscr{L}(\boldsymbol{w}^*) = 0$. The covariance $H \triangleq \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^T$ can be expressed in terms of its eigen decomposition as $\sum_{i=1}^{d}\lambda_i\boldsymbol{e}_i\boldsymbol{e}_i^T$, where d is the dimensionality of the parameter space (and the feature space) $\mathscr{H}$, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ are the eigenvalues of H, and $\{\boldsymbol{e}_1,\ldots,\boldsymbol{e}_d\}$ is the eigen-basis induced by H. In the over-parametrized setting (i.e., when $d > n$), the rank of H is at most n. Assume, w.o.l.g., that the eigenvalues are such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0 = \lambda_{k+1} = \cdots = \lambda_d$ for some $k \leq n$. We further assume that for all feature vectors $\boldsymbol{x}_i, i = 1,\ldots,n$, we have $\|\boldsymbol{x}_i\|^2 \leq \beta$. Note that this implies that the trace of H is bounded from above by β, that is, $\mathsf{tr}(H) \leq \beta$. Thus, we have $\beta > \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$. Hence, in the interpolation setting, we can write the sum of squares $\mathscr{L}(\boldsymbol{w})$ as

$$\mathscr{L}(\boldsymbol{w}) = (\boldsymbol{w}-\boldsymbol{w}^*)^T H(\boldsymbol{w}-\boldsymbol{w}^*) \tag{3}$$

For any $\boldsymbol{v} \in \mathscr{H}$, let $\mathbf{P}_{\boldsymbol{v}}$ denote the projection of $\boldsymbol{v}$ unto the subspace spanned by $\{\boldsymbol{e}_1,\ldots,\boldsymbol{e}_k\}$ and $\mathbf{Q}_{\boldsymbol{v}}$ denote the projection of $\boldsymbol{v}$ unto the subspace spanned by $\{\boldsymbol{e}_{k+1},\ldots,\boldsymbol{e}_d\}$. That is, $\boldsymbol{v} = \mathbf{P}_{\boldsymbol{v}} + \mathbf{Q}_{\boldsymbol{v}}$ is the decomposition of $\boldsymbol{v}$ into two orthogonal components: its projection onto $\mathsf{Range}(H)$ (i.e., the range space of H, which is the subspace spanned by $\{\boldsymbol{e}_1,\ldots,\boldsymbol{e}_k\}$) and its projection onto $\mathsf{Null}(H)$ (i.e., the null space of H, which is the subspace spanned by $\{\boldsymbol{e}_{k+1},\ldots,\boldsymbol{e}_d\}$). Hence, the above quadratic loss can be written as

$$\mathscr{L}(\boldsymbol{w}) = \mathbf{P}_{\boldsymbol{w}-\boldsymbol{w}^*}^T\, H\, \mathbf{P}_{\boldsymbol{w}-\boldsymbol{w}^*} \tag{4}$$

To minimize the loss in this setting, consider the following SGD update with mini-batch of size m and step size η:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta H_m(\boldsymbol{w}_t - \boldsymbol{w}^*) \tag{5}$$

where $H_m \triangleq \frac{1}{m}\sum_{i=1}^{m}\tilde{\boldsymbol{x}}_i\tilde{\boldsymbol{x}}_i^T$ is a subsample covariance corresponding to a subsample of feature vectors $\{\tilde{\boldsymbol{x}}_1,\ldots,\tilde{\boldsymbol{x}}_m\} \subset \{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}$.

Let $\boldsymbol{\delta}_t \triangleq \boldsymbol{w}_t - \boldsymbol{w}^*$. Observe that we can write (5) as

$$\mathbf{P}_{\boldsymbol{\delta}_{t+1}} + \mathbf{Q}_{\boldsymbol{\delta}_{t+1}} = \mathbf{P}_{\boldsymbol{\delta}_t} + \mathbf{Q}_{\boldsymbol{\delta}_t} - \eta H_m \left(\mathbf{P}_{\boldsymbol{\delta}_t} + \mathbf{Q}_{\boldsymbol{\delta}_t}\right) \quad (6)$$

Now, we make the following simple claim (whose proof is given in the full version of this paper).

Claim 1. *Let $\boldsymbol{u} \in \mathscr{H}$. For any subsample $\{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_m\} \subset \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, let $H_m = \frac{1}{m}\sum_{i=1}^m \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T$ be the corresponding subsample covariance matrix. Then,*

$$H_m \boldsymbol{u} \in \mathsf{Range}(H) = \mathsf{Span}\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k\}.$$

This also implies that for any $\boldsymbol{v} \in \mathsf{Null}(H) = \mathsf{Span}\{\boldsymbol{e}_{k+1}, \ldots, \boldsymbol{e}_d\}$, we must have $H_m \boldsymbol{v} = 0$.

By the above claim, the update equation (6) can be decomposed into two components:

$$\mathbf{P}_{\boldsymbol{\delta}_{t+1}} = \mathbf{P}_{\boldsymbol{\delta}_t} - \eta H_m \mathbf{P}_{\boldsymbol{\delta}_t}, \quad (7)$$

$$\mathbf{Q}_{\boldsymbol{\delta}_{t+1}} = \mathbf{Q}_{\boldsymbol{\delta}_t} \quad (8)$$

From (4), it follows that for any iteration t, the target loss function $\mathscr{L}_{\boldsymbol{w}_t}$ is not affected at all by $\mathbf{Q}_{\boldsymbol{\delta}_t}$, that is, $\mathbf{P}_{\boldsymbol{\delta}_t}$ is the only component that matters. Hence, by (7-8), we only need to consider the effective SGD update (7), i.e., the update in the span of $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k\}$.

4.1 Upper bound on the expected empirical loss

The following theorem (see proof in full paper) provides an upper bound on the expected empirical loss after t iterations of mini-batch SGD whose update step is given by (5).

Theorem 2. *For any $\lambda \in [\lambda_k, \lambda_1], m \in \mathbb{N}$, and $0 < \eta < \frac{2m}{\beta + (m-1)\lambda_1}$ define*

$$g(\lambda; m, \eta) \triangleq (1 - \eta\lambda)^2 + \frac{\eta^2 \lambda}{m}(\beta - \lambda)$$

Let $g(m, \eta) \triangleq \max_{\lambda \in [\lambda_k, \lambda_1]} g(\lambda; m, \eta)$. In the interpolation setting, for any $t \geq 1$, the mini-batch SGD with update step (5) yields the following guarantee

$$\mathbb{E}[\mathscr{L}(\boldsymbol{w}_t)] \leq \lambda_1 \cdot \mathbb{E}\left[\left\|\mathbf{P}_{\boldsymbol{\delta}_t}^2\right\|\right] \leq \lambda_1 \cdot \left(g\left(m, \eta\right)\right)^t \cdot \mathbb{E}\left[\left\|\mathbf{P}_{\boldsymbol{\delta}_0}\right\|^2\right]$$

4.2 Tightness of the bound on expected empirical loss

We now show that our upper bound given above is indeed tight in the interpolation setting for the class of quadratic loss functions defined in (3). Namely, we give a specific instance of (3) where the upper bound in Theorem 2 is tight.

Theorem 3. *There is a data set $\{(\boldsymbol{x}_i, y_i) \in \mathscr{H} \times \mathbb{R} : 1 \leq i \leq n\}$ such that the mini-batch SGD with update step (5) yields the following lower bound on the expected empirical quadratic loss $\mathscr{L}(\boldsymbol{w})$*

$$\mathbb{E}[\mathscr{L}(\boldsymbol{w}_t)] = \lambda_1 \cdot \mathbb{E}\left[\left\|\boldsymbol{\delta}_t\right\|^2\right] = \lambda_1 \cdot \left(g\left(m, \eta\right)\right)^t \cdot \mathbb{E}\left[\left\|\boldsymbol{\delta}_0\right\|^2\right]$$

See proof in the full version of this paper.

4.3 Optimal step size for a given batch size

To fully answer the first question we posed at the beginning of this section, we will derive an optimal rule for choosing the step size as a function of the batch size. Specifically, we want to find step size $\eta^*(m)$ to achieve fastest convergence. Given Theorem 2, our task reduces to finding the minimizer

$$\eta^*(m) = \arg \min_{\eta < \frac{2}{\frac{\beta}{m} + \frac{m-1}{m}\lambda_1}} g(m, \eta) \quad (9)$$

Let $g^*(m)$ denote the resulting minimum, that is, $g^*(m) = g\left(m, \eta^*(m)\right)$. The resulting expression for the minimizer $\eta^*(m)$ generally depends on the least non-zero eigenvalue λ_k of the Hessian matrix. In situations where we don't have a good estimate for this eigenvalue (which can be close to zero in practice), one would rather have a step size that is independent of λ_k. In Theorem 5, we give a near-optimal approximation for step size with no dependence on λ_k under the assumption that $\beta/\lambda_k = \Omega(n)$, which is valid in many practical settings such as in kernel learning with positive definite kernels.

We first characterize exactly the optimal step size and the resulting $g^*(m)$.

Theorem 4 (Optimal step size role as function of batch size)**.** *For every batch size m, the optimal step size function $\eta^*(m)$ and convergence rate function $g^*(m)$ are given by:*

$$\eta^*(m) = \begin{cases} \frac{m}{\beta + (m-1)\lambda_k} & m \leq \frac{\beta}{\lambda_1 - \lambda_k} + 1 \\ \frac{2m}{\beta + (m-1)(\lambda_1 + \lambda_k)} & m > \frac{\beta}{\lambda_1 - \lambda_k} + 1 \end{cases} \quad (10)$$

$$g^*(m) = \begin{cases} 1 - \frac{m\lambda_k}{\beta + (m-1)\lambda_k} & m \leq \frac{\beta}{\lambda_1 - \lambda_k} + 1 \\ 1 - 4\frac{m(m-1)\lambda_1\lambda_k}{(\beta + (m-1)(\lambda_1 + \lambda_k))^2} & m > \frac{\beta}{\lambda_1 - \lambda_k} + 1 \end{cases} \quad (11)$$

Note that if $\lambda_1 = \lambda_k$, then the first case in each expression will be valid for all $m \geq 1$.

The proof of Theorem 4 follows from the two lemmas below (whose proofs can be found in the full version of this paper).

Lemma 1. *Let $\eta_0(m) \triangleq \frac{2m}{\beta + (m-1)(\lambda_1 + \lambda_k)}$, and let $\eta_1(m) \triangleq \frac{2m}{\beta + (m-1)\lambda_1}$. Then,*

$$g(m, \eta) = \begin{cases} g^{\mathrm{I}}(m, \eta) \triangleq g(\lambda_k; m, \eta) & \eta \leq \eta_0(m) \\ g^{\mathrm{II}}(m, \eta) \triangleq g(\lambda_1; m, \eta) & \eta_0(m) < \eta \leq \eta_1(m) \end{cases}$$

Lemma 2. *Given the quantities defined in Lemma 1, let $\eta^{\mathrm{I}}(m) = \operatorname{argmin}_{\eta \leq \eta_0(m)} g^{\mathrm{I}}(m, \eta)$, and $\eta^{\mathrm{II}}(m) = \operatorname{argmin}_{\eta_0(m) < \eta \leq \eta_1(m)} g^{\mathrm{II}}(m, \eta)$. Then, we have*

1. *For all $m \geq 1$, $g^{\mathrm{I}}\left(m, \eta^{\mathrm{I}}(m)\right) \leq g^{\mathrm{II}}\left(m, \eta^{\mathrm{II}}(m)\right)$.*
2. *For all $m \geq 1$, $\eta^{\mathrm{I}}(m) = \eta^*(m)$ and $g^{\mathrm{I}}\left(m, \eta^{\mathrm{I}}(m)\right) = g^*(m)$, where $\eta^*(m)$ and $g^*(m)$ are as given by (10) and (11), respectively, (in Theorem 4).*

Proof of Theorem 4: Given Lemma 1 and item 1 of Lemma 2, it follows that $\eta^{\mathrm{I}}(m)$ is the minimizer $\eta^*(m)$ given by (9). Item 2 of Lemma 2 concludes the proof.

Nearly optimal step size with no dependence on λ_k: In practice, it is usually easy to obtain a good estimate for λ_1, but it is hard to reliably estimate λ_k which is typically much smaller than λ_1 (e.g., (Chaudhari et al., 2016)). That is why one would want to avoid dependence on λ_k in practical SGD algorithms. Under a mild assumption which is typically valid in practice, we can easily find an accurate approximation $\hat{\eta}(m)$ of optimal $\eta^*(m)$ that depends only on λ_1 and β. Namely, we assume that $\lambda_k/\beta \le 1/n$. In particular, this is always true in kernel learning with positive definite kernels, when the data points are distinct.

The following theorem provides such approximation resulting in a nearly optimal convergence rate $\hat{g}(m)$.

Theorem 5. *Suppose that $\lambda_k/\beta \le 1/n$. Let $\hat{\eta}(m)$ be defined as:*

$$\hat{\eta}(m) = \begin{cases} \frac{m}{\beta(1+(m-1)/n)} & m \le \frac{\beta}{\lambda_1-\beta/n}+1 \\ \frac{2m}{\beta+(m-1)(\lambda_1+\beta/n)} & m > \frac{\beta}{\lambda_1-\beta/n}+1 \end{cases} \tag{12}$$

Then, the step size $\hat{\eta}(m)$ yields the following upper bound on $g\left(m,\hat{\eta}(m)\right)$, denoted as $\hat{g}(m)$:

$$\hat{g}\left(m\right) = \begin{cases} 1-\frac{m\lambda_k}{\beta(1+(m-1)/n)} & m \le \frac{\beta}{\lambda_1-\beta/n}+1 \\ 1-4\frac{m(m-1)\lambda_1\lambda_k}{(\beta+(m-1)(\lambda_1+\beta/n))^2} & m > \frac{\beta}{\lambda_1-\beta/n}+1 \end{cases} \tag{13}$$

Proof. The proof follows by observing that if $\lambda_k/\beta \le 1/n$, then $\hat{\eta}(m)$ lies in the feasible region for the minimization problem in (9). In particular, $\hat{\eta}(m) \le \eta_0(m)$, where $\eta_0(m)$ is as defined in Lemma 1. The upper bound $\hat{g}\left(m\right)$ follows from substituting $\hat{\eta}(m)$ in $g^{\mathrm{I}}(m,\eta)$ defined in Lemma 1, then upper-bounding the resulting expression. □

It is easy to see that the convergence rate $\hat{g}(m)$ resulting from the step size $\hat{\eta}$ is at most factor $1+O(m/n)$ slower than the optimal rate $g^*(m)$. This factor is negligible when $m \ll n$. Since we expect $n \gg \beta$, we can further approximate $\hat{\eta}(m) \approx m/\beta$ when $m \lessapprox \beta/\lambda_1$ and $\hat{\eta} \approx \frac{2m}{\beta+(m-1)\lambda_1}$ when $m \gtrapprox \beta/\lambda_1$.

4.4 Batch size selection

In this section, we will derive the optimal batch size given a fixed computational budget in terms of the computational efficiency defined as the number of gradient computations to obtain a fixed desired accuracy. We will show that single-point batch is in fact optimal in that setting. Moreover, we will show that any mini-batch size in the range from 1 to a certain constant m^* independent of n, is nearly optimal in terms of gradient computations. Interestingly, for values beyond m^* the computational efficiency drops sharply. This result has direct implications for the batch size selection in parallel computation.

4.4.1 Optimality of a single-point batch

Suppose we are limited by a fixed number of gradient computations. Then, what would be the batch size that yields the least approximation error? Equivalently, suppose we are required to achieve a certain target accuracy ϵ (i.e., want to reach parameter $\hat{\boldsymbol{w}}$ such that $\mathscr{L}(\hat{\boldsymbol{w}}) - \mathscr{L}(\boldsymbol{w}^*) \le \epsilon$). Then, again, what would be the optimal batch size that yields the least amount of computation.

Suppose we are being charged a unit cost for each gradient computation, then it is not hard to see that the cost function we seek to minimize is $g^*(m)^{\frac{1}{m}}$, where $g^*(m)$ is as given by Theorem 4. To see this, note that for a batch size m, the number of iterations to reach a fixed desired accuracy is $t(m) = \frac{\text{constant}}{\log(1/g^*(m))}$. Hence, the computation cost is $m \cdot t(m) = \frac{\text{constant}}{\log(1/g^*(m))^{1/m}}$. Hence, minimizing the computation cost is tantamount to minimizing $g^*(m)^{1/m}$. The following theorem shows that the exact minimizer is $m = 1$. Later, we will see that *any* value for m from 2 to $\approx \beta/\lambda_1$ is actually not far from optimal. So, if we have cheap or free computation available (e.g., parallel computation), then it would make sense to choose $m \approx \beta/\lambda_1$. We will provide more details in the following subsection.

Theorem 6 (Optimal batch size under a limited computational budget)**.** *When we are charged a unit cost per gradient computation, the batch size that minimizes the overall computational cost required to achieve a fixed accuracy (i.e., maximizes the computational efficiency) is $m = 1$. Namely,*

$$\arg\min_{m\in\mathbb{N}} g^*(m)^{\frac{1}{m}} = 1$$

Here, we give a less formal but more intuitive argument based on a reasonable approximation for $g^*(m)$. Such approximation in fact is valid in most of the practical settings. In the full version of this paper, we give an exact and detailed analysis. Note that $g^*(m)$ can be written as $1 - \frac{\lambda_k}{\beta}s(m)$, where $s(m)$ is given by

$$s(m) = \begin{cases} \frac{m}{1+(m-1)\frac{\lambda_k}{\beta}} & m \le \frac{\beta}{\lambda_1-\lambda_k}+1 \\ \frac{4m(m-1)\lambda_1}{\beta\left(1+(m-1)\frac{\lambda_1+\lambda_k}{\beta}\right)^2} & m > \frac{\beta}{\lambda_1-\lambda_k}+1 \end{cases} \tag{14}$$

Proof outline: Note that $g^*(m)^{1/m} \approx e^{-s(m)/m}$. This approximation becomes very accurate when $\lambda_k \ll \lambda_1$, which is typically the case for most of the practical settings where $\lambda_1/\lambda_k \approx n$ and n is very large. Here, $s(m)$ is an approximation for the speed-up factor $t(1)/t(m)$ introduced after Theorem 1. Assuming that this approximation is accurate, for the sake of an intuitive argument, minimizing $g^*(m)^{1/m}$ becomes equivalent to maximizing $s(m)/m$. Now, note that when $m \le \frac{\beta}{\lambda_1-\lambda_k}+1$, then $s(m)/m = \frac{1}{1+(m-1)\frac{\lambda_k}{\beta}}$,

which is decreasing in m. Hence, for $m \le \frac{\beta}{\lambda_1-\lambda_k}+1$, we have $s(m)/m \le s(1) = 1$. On the other hand, when $m > \frac{\beta}{\lambda_1-\lambda_k}+1$, we have

$$\frac{s(m)}{m} = \frac{4(m-1)\lambda_1}{\beta\left(1+(m-1)\frac{\lambda_1+\lambda_k}{\beta}\right)^2},$$

which is also decreasing in m, and hence, it's upper bounded by its value at $m = m^* \triangleq \frac{\beta}{\lambda_1-\lambda_k}+1$. By direct substitution and simple cancellations, we can show that $s(m^*)/m^* \le \frac{\lambda_1-\lambda_k}{\lambda_1} < 1$. Thus, $m = 1$ is optimal.

One may wonder whether the above result is valid if the near-optimal step size $\hat{\eta}(m)$ (that does not depend on λ_k) is used. That is, one may ask whether the same optimality result is valid if the near optimal error rate function $\hat{g}(m)$ is used instead of $g^*(m)$ in Theorem 6. Indeed, we show that the same optimality remains true even if computational efficiency is measured with respect to $\hat{g}(m)$. This is formally stated in the theorem below (see proof in the full paper).

Theorem 7. *When the near-optimal step size $\hat{\eta}(m)$ is used (and assuming that $\lambda_k/\beta \le 1/n$), the batch size that minimizes the overall computational cost required to achieve a fixed accuracy is $m = 1$. Namely,*

$$\arg\min_{m\in\mathbb{N}} \hat{g}(m)^{\frac{1}{m}} = 1$$

4.4.2 Near optimal larger batch sizes

Suppose that several gradient computations can be performed in parallel. In some cases doubling the size of the mini-batch can halve the number of iterations needed to reach a fixed desired accuracy. Such observations has motivated many works to use large batch size with distributed synchronized SGD (Chen et al., 2016; Goyal et al., 2017; You et al., 2017; Smith et al., 2017). One critical problem in this large batch setting is how to choose the step size. To keep the same covariance, (Bottou et al., 2016; Li, 2017; Hoffer et al., 2017) choose the step size $\eta \sim \sqrt{m}$ for batch size m. On the other hand, (Krizhevsky, 2014; Goyal et al., 2017; You et al., 2017; Smith et al., 2017) observed that rescaling the step size $\eta \sim m$ works well in practice when m is not too large. To explain these observations, we directly connect the parallelism, or the batch size m, to the required number of iterations $t(m)$ defined previously. It turns out that (a) when the batch size is small, doubling the size will almost halve the required iterations; (b) after the batch size surpasses certain value, increasing the size to any amount would only reduce the required iterations by at most a constant factor.

Our analysis uses the optimal step size and convergence rate derived in Theorem 4. Now consider the factor by which we save the number of iterations when increasing the batch size from 1 to m. Using the approximation $g^*(m)^{\frac{1}{m}} \approx e^{-\frac{\lambda_k}{\beta}\cdot\frac{s(m)}{m}}$, we have $s(m) \approx \frac{t(1)}{t(m)}$, the speedup factor. The change of $s(m)$ is illustrated in Figure 2 where two regimes are highlighted:

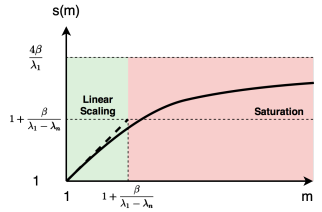


Figure 2: Factor of iterations saved: $\frac{t(1)}{t(m)} \approx s(m)$

Linear scaling regime ($m \le \frac{\beta}{\lambda_1-\lambda_k}+1$): This is the regime where increasing the batch size m will quickly drive down $t(m)$ needed to reach certain accuracy. When $\lambda_k \ll \lambda_1$, $s(m) \approx m$, which suggests $t(m/2) \approx 2\cdot t(m)$. In other words, doubling the batch size in this regime will roughly halve the number of iterations needed. Note that we choose step size $\eta \leftarrow \frac{m}{\beta+(m-1)\lambda_k}$. When $\lambda_k \le \frac{\beta}{n} \ll \lambda_1$, $\eta \sim m$, which is consistent with the linear scaling heuristic used in (Krizhevsky, 2014; Goyal et al., 2017; Smith et al., 2017). In this case, the largest batch size in the linear scaling regime can be practically calculated through

$$m^* = \frac{\beta}{\lambda_1-\lambda_k}+1 \approx \frac{\beta}{\lambda_1-\beta/n}+1 \approx \frac{\beta}{\lambda_1}+1 \quad (15)$$

Saturation regime ($m > \frac{\beta}{\lambda_1-\lambda_k}+1$): Increasing batch size in this regime becomes much less beneficial. Although $s(m)$ is monotonically increasing, it is upper bounded by $\lim_{m\to\infty} s(m) = \frac{4\beta}{\lambda_1}$. In fact, since $t(m^*)/\lim_{m\to\infty} t(m) < 4$ for small λ_k, no batch size in this regime can reduce the needed iterations by a factor of more than 4.

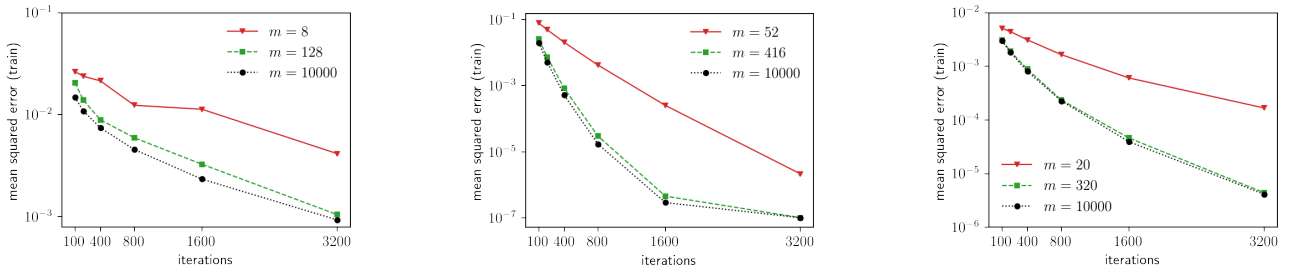
5 Experimental Results

This section will provide empirical evidence for our theoretical results on the effectiveness of mini-batch SGD in the interpolated setting. We first consider a kernel learning problem, where the parameters β, λ_1, and m^* can be computed efficiently (see (Ma & Belkin, 2017) for details). In all experiments we set the step size to be $\hat{\eta}$ defined in (12).

Remark: near optimality of $\hat{\eta}$ in practice. We observe empirically that increasing the step size from $\hat{\eta}$ to $2\,\hat{\eta}$ consistently leads to divergence, indicating that $\hat{\eta}$ differs from the optimal step size by at most a factor of 2. This is consistent with our Theorem 5 on near-optimal step size.

5.1 Comparison of SGD with critical mini-batch size m^* to full gradient descent

Theorem 4 suggests that SGD using batch size m^* defined in (15) can reach the same error as GD using at most 4 times the number of iterations. This is consistent with our experimental results for MNIST (LeCun et al., 1998), HINTS (Healy et al., 2013), and TIMIT (Garofolo et al., 1993)

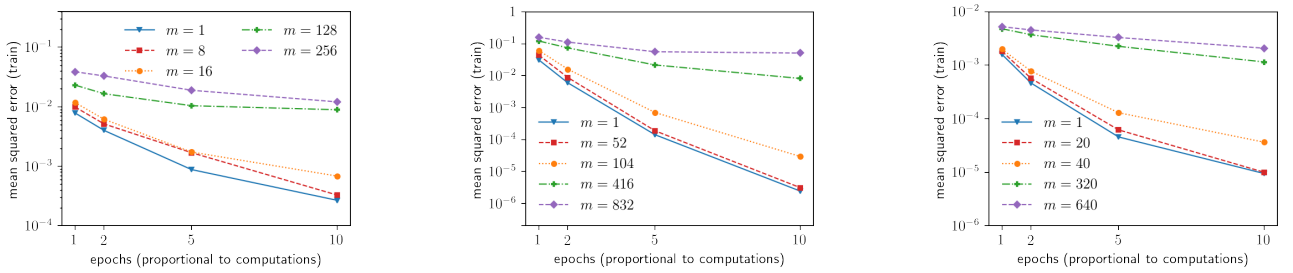


(a) MNIST (Gaussian, $\sigma = 5$), $\beta = 1, \lambda_1 = 0.15, m^* \approx 8$

(b) HINT-S (EigenPro-Laplace, $\sigma = 20$), $\beta = 0.6, \lambda_1 = 0.012, m^* \approx 52$

(c) TIMIT (Gaussian, $\sigma = 11$), $\beta = 1, \lambda_1 = 0.054, m^* \approx 20$

Figure 3: Comparison of training error ($n = 10^4$) for different mini-batch sizes (m) vs. number of iterations



(a) MNIST (Gaussian, $\sigma = 5$), $\beta = 1, \lambda_1 = 0.15, m^* \approx 8$

(b) HINT-S (EigenPro-Laplace, $\sigma = 20$), $\beta = 0.6, \lambda_1 = 0.012, m^* \approx 52$

(c) TIMIT (Gaussian, $\sigma = 11$), $\beta = 1, \lambda_1 = 0.054, m^* \approx 20$

Figure 4: Comparison of training error ($n = 10^4$) for different mini-batch sizes (m) vs. number of epochs (proportional to computation, note for n data points, $n \cdot N_{epoch} = m \cdot N_{iter}$)

in Figure 3. Moreover, in line with our analysis, SGD with batch size larger than m^* but still much smaller than the data size, converges nearly identically to full gradient descent.

Remark. Since our analysis is concerned with the training error, only the training error is reported here. For completeness, we report the test error in the full version of this paper. As consistently observed in such over-parametrized settings, test error decreases with the training error.

5.2 Optimality of batch size $m = 1$

Our theoretical results, Theorem 6 and Theorem 7 show that $m = 1$ achieves the optimal computational efficiency. Note for a given batch size, the corresponding optimal step size is chosen according to equation (12). The experiments in Figure 4 show that $m = 1$ indeed achieves the lowest error for any fixed number of epochs.

5.3 Linear scaling and saturation regimes

In the interpolation regime, Theorem 6 shows linear scaling for mini-batch sizes up to a (typically small) "critical" batch size m^* defined in (15) followed by the saturation regime. In Figure 4 we plot the training error for different batch sizes as a function of the number of epochs. Note that the number of epochs is proportional to the amount of computation measured in terms of gradient evaluations. The linear scaling regime ($1 \leq m \leq m^*$) is reflected in the small difference in the training error for $m = 1$ and $m = m^*$ in Figure 4 (the bottom three curves. As expected from our theoretical results, they have similar computational efficiency. On the other hand, we see that large mini-batch sizes ($m \gg m^*$) require drastically more computations, which is the saturation phenomenon reflected in the top two curves.

Relation to the "linear scaling rule" in neural networks. A number of recent large scale neural network methods including (Krizhevsky, 2014; Chen et al., 2016; Goyal et al., 2017) use the "linear scaling rule" to accelerate training using parallel computation. After the initial "warmup" stage to find a good region of parameters, this rule suggest increasing the step size to a level proportional to the mini-batch size m. In spite of the wide adoption and effectiveness of this technique, there has been no satisfactory explanation (Krizhevsky, 2014) as the usual variance-based analysis suggests increasing the step size by a factor of $\sqrt{m}$ instead of m (Bottou et al., 2016). We note that this "linear scaling" can be explained by our analysis, assuming that the warmup stage ends up in a neighborhood of an interpolating minimum.

Acknowledgements

We used a Titan Xp GPU provided by Nvidia for the experiments. We thank NSF for financial support.

References

Allen-Zhu, Z. Katyusha: The first direct acceleration of stochastic gradient methods. *arXiv preprint arXiv:1603.05953*, 2016.

Bartlett, P., Foster, D. J., and Telgarsky, M. Spectrally-normalized margin bounds for neural networks. In *NIPS*, 2017.

Belkin, M., Ma, S., and Mandal, S. To understand deep learning we need to understand kernel learning. *arXiv preprint arXiv:1802.01396*, 2018.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Canziani, A., Paszke, A., and Culurciello, E. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

Chaudhari, P., Choromanska, A., Soatto, S., and LeCun, Y. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

Chen, J., Monga, R., Bengio, S., and Jozefowicz, R. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Defazio, A., Bach, F., and Lacoste-Julien, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, 2014.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett, D. S. Darpa timit acoustic-phonetic continous speech corpus cd-rom. *NIST speech disc*, 1-1.1, 1993.

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 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *ICML*, 2015.

Healy, E. W., Yoho, S. E., Wang, Y., and Wang, D. An algorithm to improve speech recognition in noise for hearing-impaired listeners. *The Journal of the Acoustical Society of America*, 134(4), 2013.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprint arXiv:1705.08741*, 2017.

Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Jain, P., Kakade, S. M., Kidambi, R., Netrapalli, P., and Sidford, A. Parallelizing stochastic approximation through mini-batching and tail-averaging. *arXiv preprint arXiv:1610.03774*, 2016.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

Kaczmarz, S. Angenaherte auflosung von systemen linearer gleichungen. *Bull. Int. Acad. Sci. Pologne, A*, 35, 1937.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, 1998.

Li, M. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, 2017.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *KDD*, 2014.

Ma, S. and Belkin, M. Diving into the shallows: a computational perspective on large-scale shallow learning. *arXiv preprint arXiv:1703.10622*, 2017.

Moulines, E. and Bach, F. R. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *NIPS*, 2011.

Needell, D., Ward, R., and Srebro, N. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *NIPS*, 2014.

Roux, N. L., Schmidt, M., and Bach, F. R. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

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

Salakhutdinov, R. Deep learning tutorial at the Simons Institute, Berkeley, 2017. URL https://simons.berkeley.edu/talks/ruslan-salakhutdinov-01-26-2017-1.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Statist.*, 26(5), 1998.

Smith, S. L., Kindermans, P.-J., and Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Strohmer, T. and Vershynin, R. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2), 2009.

Takác, M., Bijral, A. S., Richtárik, P., and Srebro, N. Mini-batch primal and dual methods for svms. In *ICML*, 2013.

Xiao, L. and Zhang, T. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4), 2014.

Yin, D., Pananjady, A., Lam, M., Papailiopoulos, D., Ramchandran, K., and Bartlett, P. Gradient diversity: a key

ingredient for scalable distributed learning. In *AISTATS*, 2018.

You, Y., Gitman, I., and Ginsburg, B. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.