École Polytechnique Fédérale de Lausanne

On SGD with Momentum

by Maximilian Plattner

Master Thesis

Approved by the Examining Committee:

Dr. sc. ETH Sebastian Stich
Thesis Supervisor

Matteo Pagliardini, MSc.
Thesis Supervisor

Univ.Prof. Dipl.-Ing.(BA) Dr.rer.nat. Thomas Gärtner
Thesis Supervisor

Univ. Prof. Dr. sc. ETH Martin Jaggi
Thesis Supervisor

EPFL IC IINFCOM MLO
[redacted]

June 28, 2022

Acknowledgements

I would like to thank Sebastian Stich, Matteo Pagliardini and Thomas Gärtner for their time and support. The helpful conversations and inspiring discussions with them have made writing this work an exciting journey into the world of machine learning research.

Lausanne, June 28, 2022 Maximilian Plattner

Abstract

Stochastic Gradient Descent (SGD) is the workhorse for training large-scale machine learning applications. Although the convergence rate of its deterministic counterpart, Gradient Descent (GD), can be shown to be accelerated by adaptations that use the notion of momentum, e.g., Heavy Ball (HB) or Nesterov Accelerated Gradient (NAG), the theory could not prove, by means of local convergence analysis, that such modifications provide faster convergence rates in the stochastic setting. This work empirically establishes that a positive momentum coefficient in SGD has the effect of enlarging the algorithm's learning rate, not contributing to a boost in performance per se. For the deep learning setting, however, this enlargement tends to be conducted in a way robust to unfavorable initialization points. Given these findings, this work derives a heuristic, the *Momentum Linear Scaling Rule (MLSR)*, to transfer from a small-batch setting to a large-batch setting in deep learning while approximately maintaining the same generalization performance.

Contents

Introduction

We are in an exciting area of machine learning research in which the increasing data and model scale is swiftly improving accuracy in computer vision [42], natural language processing [2] and speech recognition [54]. The workhorse for training these large-scale machine learning applications is Stochastic Gradient Descent (SGD) [44]. For its non-stochastic counterpart, Gradient Descent (GD) [3], there exist adaptations, e.g., Heavy Ball (HB) [41] or Nesterov Accelerated Gradient (NAG) [36], which use the notion of momentum, capable to provably accelerate convergence. In the stochastic setting, however, the theory could not identify [62, 30, 47, 26, 61], with the tools of local convergence analysis, an increase in training speed when employing SGD with momentum (SGDM). In spite of the absence of such theoretical guarantees, SGDM is a popular [22, 39, 33, 26] optimizer in machine learning and said to empirically [51, 26] boost the training performance. In this work, we will empirically establish that a momentum coefficient in the stochastic setting generally has the effect of merely enlarging the learning rate of SGD, not contributing to faster training per se. For deep learning applications, however, this enlargement is conducted in a fashion robust to unfavorable initialization points. Given our findings, we will derive a promising heuristic, the *Momentum Linear Scaling Rule (MLSR)*, to transfer from a small-batch setting to a large-batch setting in deep learning while approximately maintaining the same generalization performance. Chapter 1 introduces important concepts of mathematical optimization used in machine learning as well as their relation to generalization efficiency. Chapter 2 discusses the usage of momentum in SGD on the basis of Stochastic Heavy Ball (SHB) [41] and Stochastic Nesterov Accelerated Gradient (SNAG) [36]. Chapter 3 presents the contributions of this work with chapter 4 summarizing our conclusions.

Chapter 1

Optimization and machine learning

Many real-world problems can be viewed as optimization problems that aim to minimize a loss function or maximize a reward function[1]. In that regard, machine learning techniques rely heavily on the field of mathematical optimization and the efficiency of respective numerical methods. In this chapter, we give a brief introduction to optimization, its use in machine learning and its relation to generalization.

1.1 Notation, definitions and assumptions

In this section we introduce some general notations, definitions and assumptions which are standard in the context of optimization.

1.1.1 Notation

We denote vectors by lowercase letters in bold face, matrices by uppercase letters in bold face and scalars by lowercase letters, e.g. x_i is a scalar representing the ith coordinate of the vector $\boldsymbol{x}$ and $\boldsymbol{A}$ is a matrix. We consider twice-differentiable functions $f : \mathbb{R}^p \to \mathbb{R}$, for which we denote by $\boldsymbol{x}^\star$ a minimizer of f, i.e., $\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$ and by $\nabla f(\boldsymbol{x}) \coloneqq \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ the gradient of f at $\boldsymbol{x} \in \mathbb{R}^p$. The Hessian of f at a point $\boldsymbol{x} \in \mathbb{R}^p$ is denoted by $\nabla^2 f(\boldsymbol{x}) \coloneqq \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$. Furthermore, we use the convention that $\mathcal{O}(\cdot)$ hides constants.

[1]Note that, given a function $f(x)$, it always holds that $\max_x f(x) = -(\min_x -f(x))$. W.l.o.g., one can hence restrict their attention to minimization problems.

1.1.2 Definitions

A reoccurring theme in optimization is the notion of convexity which gives us a guarantee that every local minimum is a global one.

Definition 1 (convexity [37]). *A differentiable function $f : \mathbb{R}^p \to \mathbb{R}$ is convex if and only if*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p. \tag{1.1}$$

An even stronger version of convexity is the notion of strong convexity which gives us the additional guarantee that our objective function is lower-bounded by a quadratic function.

Definition 2 (μ-strong convexity [37]). *A differentiable function $f : \mathbb{R}^p \to \mathbb{R}$ is μ-strongly convex with $\mu > 0$ if*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle + \frac{\mu}{2} \|\boldsymbol{y} - \boldsymbol{x}\|^2 \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p. \tag{1.2}$$

A convenient property of an objective function is that its gradients do not change arbitrarily fast within a neighborhood which can be formalized with the notion of smoothness.

Definition 3 (L-smoothness [37]). *A differentiable function $f : \mathbb{R}^p \to \mathbb{R}$ is L-smooth if its gradient is L-Lipschitz with $L > 0$, i.e.,*

$$\|\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\| \leq L \|\boldsymbol{y} - \boldsymbol{x}\| \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p. \tag{1.3}$$

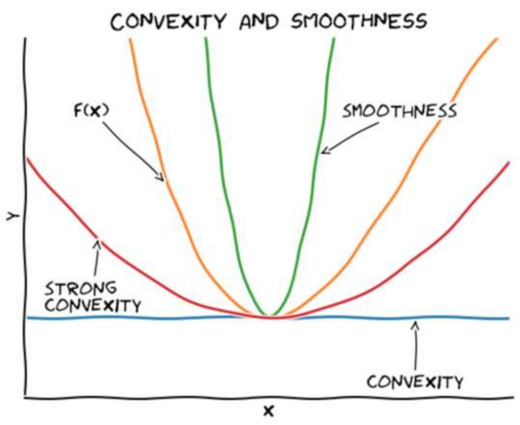


Figure 1.1: An illustration [40] of definitions 1-3.

1.1.3 Assumptions

To theoretically analyze the local convergence behavior of stochastic first-order methods such as SGD, it is convenient to do this under assumptions. We list the three most prominent assumptions used in state-of-the-art [30, 62, 26, 1] convergence analysis in the following.

Assumption 1 (Unbiased gradient oracles). *Almost surely, we have*

$$\mathbb{E}[g(\boldsymbol{x}_t)|\xi_t] = \nabla f(\boldsymbol{x}_t), \tag{1.4}$$

for each iteration $t = 1, ..., T$ with ξ_t a realization of a random seed.

Assumption 2 (Independent realizations). *The realizations $\{\xi_t\}_{t=1}^{T}$, with T the number of iterations, are independent.*

Assumption 3 (Bounded variance). *There exists $\sigma > 0$ such that*

$$\mathbb{E}[\|g(\boldsymbol{x}_t) - \nabla f(\boldsymbol{x}_t)\|^2] \leq \sigma^2 \tag{1.5}$$

for each iteration $t = 1, ..., T$.

1.2 Optimization in machine learning

Our objective [1] in supervised learning[2] is to determine a prediction function $h : \mathcal{A} \to \mathcal{B}$ such that, given $\boldsymbol{a} \in \mathcal{A}$, the value $h(\boldsymbol{a})$ offers an accurate prediction about the true output $b \in \mathcal{B}$. We assume that the prediction function h has a fixed form and is parameterized by a vector $\boldsymbol{x} \in \mathbb{R}^p$ over which the optimization is to be performed. That is, for some $h(\cdot;\cdot) : \mathcal{A} \times \mathbb{R}^p \to \mathcal{B}$, we consider the family of prediction functions

$$\mathcal{H} = \{h(\cdot;\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^p\}.$$

We aim to find the prediction function $h \in \mathcal{H}$ that minimizes the loss incurred from inaccurate predictions. That is, we assume a given loss function $\ell : \mathcal{B} \times \mathcal{B} \to \mathbb{R}$ which, given an input-output pair $(\boldsymbol{a}, b)$, yields the loss $\ell(h(\boldsymbol{a};\boldsymbol{x}), b)$ where $h(\boldsymbol{a};\boldsymbol{x})$ and b are the predicted and true outputs, respectively. Having access to a set of realizations $\{\xi_i\}_{i=1}^n$ of a random seed ξ corresponding to $n \in \mathbb{N}$ drawn i.i.d. input-output samples $\{(\boldsymbol{a}_i, b_i)\}_{i=1}^n \subseteq \mathcal{A} \times \mathcal{B}$, one seeks to minimize the empirical risk function $f : \mathbb{R}^p \to \mathbb{R}$, given as

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(\boldsymbol{a}_i;\boldsymbol{x}), b_i) = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x};\xi_i) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}), \tag{1.6}$$

where f denotes the composition of the loss function ℓ and the prediction function h. Below we briefly discuss two prominent examples of how f can be formulated.

Logistic regression

Logistic regression [16] is a classic approach to binary classification where we model the conditional probability of observing a class label $b_i \in \{-1, 1\}$ given a set of features $\boldsymbol{a}_i \in \mathbb{R}^p$. The prediction function h is given by

$$h(\boldsymbol{a}_i;\boldsymbol{x}) \coloneqq s(\boldsymbol{a}_i^T \boldsymbol{x}) \tag{1.7}$$

with $s(u) = \frac{1}{1+\exp(-u)}$ the standard logistic function. Following the principle of maximum likelihood estimation, ℓ corresponds to the binary cross-entropy loss and minimizing (1.6) accordingly yields a convex[3] problem with $\boldsymbol{x}^\star$ being a linear classifier.

[2]Narrowing down the discussion to supervised learning is reasonable as many unsupervised and other learning techniques reduce to optimization problems of comparable form. [53]

[3]Note that employing L_2-regularization with regularization parameter μ yields a μ-strongly convex problem.

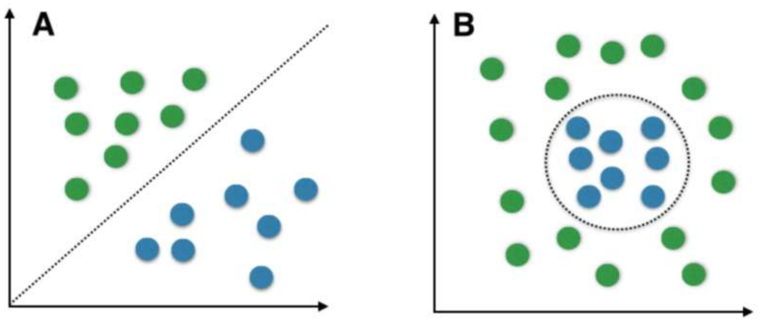


Figure 1.2: An illustration [43] of linearly separable (left) and non-linearly separable (right) data.

Deep neural networks

Modern machine learning applications often require learning algorithms to handle non-linearly separable data in very high-dimensional spaces. Thus, a theoretically [13] as well as practically [29] promising alternative to linear classifiers such as logistic regression are Deep Neural Networks (DNN). DNNs [1] describe the prediction function h by applying successive transformations, made in M layers, to a given input vector $\boldsymbol{a}_i \in \mathbb{R}^{d_1}$, where d_m is the output dimension of the mth transformation. A canonical fully connected layer performs the computation

$$\boldsymbol{a}_i^{(m)} = s(\boldsymbol{X}^{(m)}\boldsymbol{a}_i^{(m-1)} + \mu^{(m)}) \in \mathbb{R}^{d_m}, \tag{1.8}$$

where $\boldsymbol{a}_i^{(1)} = \boldsymbol{a}_i$, the matrix $\boldsymbol{X}^{(m)} \in \mathbb{R}^{d_m \times d_{m-1}}$ and vector $\mu^{(m)} \in \mathbb{R}^m$ contain the parameters of the mth layer, and $s(\cdot)$ is a component-wise non-linear activation function. As such, we have $h(\boldsymbol{a}_i; \boldsymbol{x}) := \boldsymbol{a}_i^{(M)}$, where the parameter vector $\boldsymbol{x}$ collects the parameters $\{(\boldsymbol{X}^{(m)}, \mu^{(m)})\}_{m=1}^{M}$ of the successive layers. Independent of the loss function ℓ, minimizing (1.6) accordingly generally leads to a highly non-linear and non-convex optimization problem, making it intractable [15] to solve to global optimality.

1.3 Convex and non-convex optimization

The examples in section 1.2 illustrate the variety [1] of optimization problems that arise in machine learning: one comprises convex optimization problems which are derived from the usage of logistic regression while the other one involves highly non-linear and non-convex problems which are derived from the usage of deep neural networks. This section gives a brief overview of convex and non-convex optimization. As in (1.6), we consider the finite sum minimization problem where, given n functions $f_i : \mathbb{R}^p \to \mathbb{R}$ for $i = 1, ..., n$, we want to find $\boldsymbol{x}^\star$ that minimizes the average function value. That is, formally, we seek to find $\boldsymbol{x}^\star$ such that

$$\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}), \qquad f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}). \tag{1.9}$$

1.3.1 Convex optimization

In the convex setting, if the derivative of the function $f(\boldsymbol{x})$ is defined everywhere on its domain $\mathbb{R}^p$ and can be inverted, a minimizer $\boldsymbol{x}^\star$ is obtained by the first-order constraint $\nabla f(\boldsymbol{x}) = \mathbf{0}$. However, if there exists no such closed form solution for $\boldsymbol{x}^\star$, as is the case with many machine learning applications such as logistic regression, we need to revert to numerical algorithms to search for a minimizer.

Gradient Descent (GD)

A natural choice for the search direction is the negative gradient, i.e., $-\nabla f(\boldsymbol{x})$, and the respective method is fittingly dubbed Gradient Descent (GD) [3]. GD is an iterative method which, in each iteration $t = 1, ..., T$, takes a step with learning rate $\alpha \in \mathbb{R}_{++}$ towards the opposite direction of the gradient computed at the current point $\boldsymbol{x}_t$. Note that the gradient is only a local indication of the direction with the steepest slope and hence the learning rate α should be carefully chosen in order to ensure convergence to a neighborhood of $\boldsymbol{x}^\star$.

learning rate α

Stochastic Gradient Descent (SGD)

Computing the average gradient over n functions f_i can be expensive if n is large. This is particularly true for large-scale machine learning applications [1] where n represents the number of data points in a dataset with typically $n > 10^6$. Instead, we can fall back to

Stochastic Gradient Descent (SGD) [44], where in each iteration $t = 1, ..., T$, we sample uniformly at random B distinct indices $\{i_{t1}, i_{t2}, ..., i_{tB}\} = \mathcal{B}_t$ from $[n]$, where $\mathcal{B}_t$ denotes the batch [10] at iteration t and $B = |\mathcal{B}_t|$ denotes the batch size. We then apply a step with learning rate α in the opposite direction of $g(\boldsymbol{x}_t) \coloneqq \frac{1}{B}\sum_{i\in\mathcal{B}_t} \nabla f_i(\boldsymbol{x}_t)$, where $g(\boldsymbol{x}_t)$ denotes the stochastic gradient at $\boldsymbol{x}_t$. In fact, $g(\boldsymbol{x}_t)$ is an unbiased[4] estimator of the true gradient, i.e., $\mathbb{E}[g(\boldsymbol{x}_t)] = \nabla f(\boldsymbol{x}_t)$, and one iteration of SGD is n/B times cheaper than one iteration of GD.

batch size B

Algorithm 1 Stochastic Gradient Descent

```
Require: initial $\boldsymbol{x}_1 \in \mathbb{R}^p$, iterations $T$, learning rate $\alpha \in \mathbb{R}_{++}$
1: for $t = 1, ..., T$ do
2:     $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \alpha g(\boldsymbol{x}_t)$          ▷ SGD step
3: end for
4: return $\boldsymbol{x}_T$                                    ▷ return last iterate
```

Figure 1.3: We can describe both GD and SGD within the framework of SGD where the special case of $g(\boldsymbol{x}_t) = \nabla f(\boldsymbol{x}_t)$ for all $t = 1, ..., T$ or $B = n$ corresponds to GD.

1.3.2 Non-convex optimization

The global optimization of non-convex objectives such as deep neural networks is an NP-hard problem in general. Hence, rather than trying to find $\boldsymbol{x}^\star$ in (1.9), one usually contents [15] oneself with a local minimum $\bar{\boldsymbol{x}}$. For a twice-differentiable function $f : \mathbb{R}^p \to \mathbb{R}$, we call a point $\bar{\boldsymbol{x}} \in \mathbb{R}^p$ a local minimum if $\nabla f(\bar{\boldsymbol{x}}) = \boldsymbol{0}$ and all eigenvalues in $\nabla^2 f(\bar{\boldsymbol{x}})$ are positive. In the case of deep learning, we usually [15] search for local minima using SGD.

local minimum

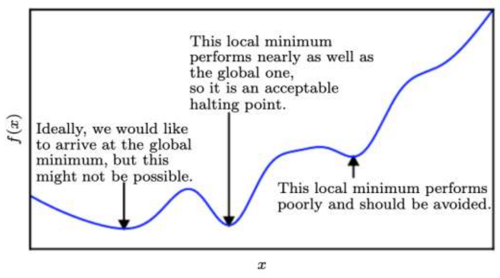


Figure 1.4: An illustration [9] of a non-convex landscape.

[4]This implies that assumption 1 is satisfied.

1.3.3 Local convergence analysis

To theoretically analyze the "speed" at which SGD reaches a specific region of the objective function, e.g., a neighborhood of a local minimum $\bar{\boldsymbol{x}}$, one can make use of local convergence analysis. For instance, if we fix a constant learning rate α in SGD for the iterations $t = 1, ...T$, we are confronted [1] under assumptions 1-3 for a smooth non-convex objective f with a local convergence rate of

$$\underbrace{\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{x}_t)\|^2\right]}_{\text{current distance}} \leq \mathcal{O}\left(\underbrace{\frac{f(\boldsymbol{x}_1) - f(\bar{\boldsymbol{x}})}{\alpha T}}_{\text{optimization term}} + \underbrace{\alpha\sigma^2}_{\text{noise term}}\right) \leq \varepsilon, \tag{1.10}$$

where $\boldsymbol{x}_1$ is the initialization point of SGD, $\bar{\boldsymbol{x}}$ denotes a local minimum and ε is a tolerance level. Without going into specifics, we see in (1.10) the general [1] framework with respect to the local convergence rate of SGD for a variety [5] of objectives. That is, one usually describes an upper-bound on the *current distance* to the desired region by an *optimization term*, comprising the initial distance from $\boldsymbol{x}_1$ to said region, as well as a *noise term* which quantifies the stochasticity in the gradients. Indeed, we need α and T to be large in order to reduce the optimization term while we need α to be small in order to decrease the noise term. As a compromise, one usually [9, 1] decays α over the course of the iterations. Note that the noise term becomes a more dominant part of the convergence rate as one increases the variance upper bound, $\sigma^2 \to \infty$, or decreases the tolerance level, $\varepsilon \to 0$.

[5] In particular, this framework captures [1] the local convergence behavior of SGD for the objective functions we will consider in chapter 3, i.e., smooth (strongly) convex and smooth non-convex problems. Note that for strongly convex objectives, the optimization term is even reduced linearly with $T \to \infty$.

1.4 Generalization

The process of optimizing the empirical risk function (1.6) in the context of machine learning is also referred to as *training* a predictor. In fact, the practical goal of this training process is to have the predictor perform well on unseen data, that is, to obtain a predictor which *generalizes* well. We simulate this unseen data by holding out a portion of the dataset which is referred to as the test set. However, minimizing the empirical risk can lead to overfitting, i.e., the model fits closely to the training set but does not generalize well to unseen data. In the convex setting, we are able to counteract overfitting through regularization. Regularization is an approach to compromise between an accurate solution to the empirical risk minimization problem and the solution's complexity by introducing a penalty term. For instance, we can regularize a logistic regression problem with the L_2-norm which renders the final training objective strongly convex.[5]

However, to mitigate overfitting in deep learning problems, we are required not only to find a local minimum within the non-convex landscape but to avoid *sharp* local minima during our search. For a twice-differentiable function f, a local minimum $\bar{\boldsymbol{x}}$ is characterized as sharp [17, 35] if there exist numerous large positive eigenvalues in $\nabla^2 f(\bar{\boldsymbol{x}})$. These sharp minima tend to generalize [17] less well than flat minima at which the Hessian exhibits several small positive eigenvalues.

sharp local minimum

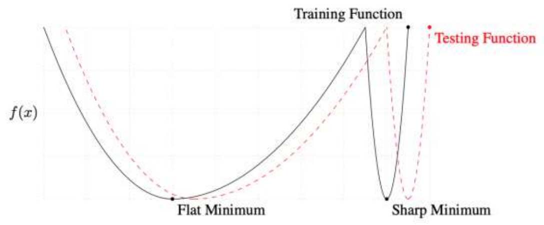


Figure 1.5: An illustration [17] of a flat and sharp local minimum in the context of generalization for deep learning.

SGD is the main optimizer for the training of deep neural networks [15, 9] and, from a theoretical point of view [57], this algorithm exponentially depends on stochastic gradient noise (SGN) — which is considered [32] to be proportional to $\frac{\alpha}{B}$ — to escape sharp minima.

SGN $\frac{\alpha}{B}$

1.4.1 Large-batch training and the Linear Scaling Rule (LSR)

Large-batch training, i.e., $B \to n$, is able to efficiently utilize [10] parallel computation to accelerate the training process of large-scale machine learning applications. However, to achieve good generalization in deep learning, we are required [32, 57] to have $\frac{\alpha}{B}$ sufficiently large in order to escape sharp minima when training with SGD. Hence, when transferring from a small-batch setting with batch size B to a large-batch setting with batch size cB, where we scale the original batch size B by $c > 1$, a natural heuristic to maintain a sufficient level of SGN is to scale the learning rate α of SGD by the same factor c. This heuristic [10] is called the *Linear Scaling Rule (LSR)*. However, the literature [10, 34, 57, 27] suggests that an increase of the learning rate α only works in a narrow range since too large initial learning rates may lead to optimization divergence or bad convergence. To mitigate optimization challenges in the early stages of convergence when applying the LSR, the literature [10] has proposed various warmup strategies with respect to the learning rate α.

Chapter 2

Momentum in SGD

It has been found that the convergence rate of GD, i.e., the special case of SGD employing *deterministic* gradients, can be *provably* [8] accelerated through adaptations of the algorithm's update step. The most prominent extensions of GD that achieve such acceleration are Polyak's Heavy Ball (HB) method [41] and Nesterov's Accelerated Gradient (NAG) method [36] and we refer the interested reader to Ghadimi et al. [8] for an in-depth discussion of the convergence rates of these methods in the deterministic setting.

As outlined in sections 1.3 and 1.4, SGD has practical advantages over GD with regard to computing efficiency and, in the case of deep learning, generalization efficiency. However, convergence analysis [62, 30, 47, 26, 61, 9] suggests that the Stochastic HB (SHB) method and the Stochastic NAG (SNAG) method have no theoretical advantage over SGD. In spite of that, both methods are still popular [22, 51] extensions to SGD in practice. This chapter briefly introduces SHB and SNAG and identifies an open question with regard to the benefit of these methods for stochastic gradients.

2.1 Stochastic Heavy Ball (SHB)

SHB [41] is the method to which the literature generally [9, 61, 60, 62] refers as *SGD with momentum (SGDM)*. There are a number of formulations[1] of SHB discussed and analyzed in the literature and we introduce the most popular[2] one in Algorithm 2.

Algorithm 2 Stochastic Heavy Ball (SHB)

```
Require: initial $\boldsymbol{x}_1 \in \mathbb{R}^p$, iterations $T$, learning rate $\alpha \in \mathbb{R}_{++}$, momentum $\beta \in [0,1)$
1: $\boldsymbol{m}_0 \leftarrow \mathbf{0}$                                    ▷ set initial momentum term to 0
2: for $t = 1, ..., T$ do
3:     $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + g(\boldsymbol{x}_t)$        ▷ compute momentum term
4:     $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \alpha \boldsymbol{m}_t$             ▷ update step
5: end for
6: return $\boldsymbol{x}_T$                                           ▷ return last iterate
```

Let us first compare the SHB algorithm with SGD. We observe that we introduce an auxiliary sequence $\boldsymbol{m}_t$, the momentum term, which accumulates past stochastic gradients $g(\boldsymbol{x}_i), i = 1, ..., t$. We then update our current iterate $\boldsymbol{x}_t$ in the negative direction of $\boldsymbol{m}_t$ with a learning rate α. In fact, one can rewrite the momentum term as

$$\boldsymbol{m}_t = \sum_{i=1}^{t} \beta^{t-i} g(\boldsymbol{x}_i), \tag{2.1}$$

where $\beta \in [0,1)$ denotes the momentum coefficient. Indeed, it is easy to see that when we set $\beta = 0$, we recover SGD.

momentum β

2.1.1 On an effective learning rate

By observing Equation (2.1), we notice that $\boldsymbol{m}_t$ is the largest when successive stochastic gradients point in exactly the same direction. In fact, if SGDM always observes a stochastic gradient $\boldsymbol{g}$, the momentum term $\boldsymbol{m}_t$ corresponds to $\frac{1}{(1-\beta)}\boldsymbol{g}$ in the long-time limit. Therefore, it is convenient [9, 59, 27] to think of SHB as SGD with an *effective learning rate* of

effective learning rate $\tilde{\alpha} = \frac{\alpha}{1-\beta}$

$$\tilde{\alpha} = \frac{\alpha}{1-\beta}. \tag{2.2}$$

[1]Please refer to Appendix .1 for a discussion of several formulations of SHB and SNAG introduced in the literature.
[2]Note that the SGD optimizer in both PyTorch [39] and TensorFlow [33] corresponds to Algorithm 2 with $\beta = 0$ as the default parameter setting.

Note that $\tilde{\alpha}$ constitutes an approximation but acknowledges the literature's claim that SHB with $\beta > 0$ changes [59] or, more precisely, enlarges [49] the learning rate α of SGD.

2.2 Stochastic Nesterov Accelerated Gradient (SNAG)

SNAG [36], in its original version, is not a momentum method, however, Sutskever et al. [51] derived a relation to SHB which is also dubbed *Nesterov momentum* [9] and given in Algorithm 3.

Algorithm 3 Stochastic Nesterov Accelerated Gradient (SNAG)

```
Require: initial $\boldsymbol{x}_1 \in \mathbb{R}^p$, iterations $T$, learning rate $\alpha \in \mathbb{R}_{++}$, momentum $\beta \in [0, 1)$
1: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$                                   ▷ set initial momentum to 0
2: for $t = 1, ..., T$ do
3:     $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} - \alpha g(\boldsymbol{x}_t + \beta \boldsymbol{m}_{t-1})$    ▷ compute Nesterov momentum
4:     $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{m}_t$                      ▷ update step
5: end for
6: return $\boldsymbol{x}_T$                                      ▷ return last iterate
```

2.3 What is the benefit of momentum in SGD?

In spite of the provable [8] acceleration capacity in the deterministic setting, the literature [62, 30, 47, 26, 61, 9] could not identify — with the tools of local convergence analysis — any theoretical advantage of SHB and SNAG over plain SGD in the stochastic setting. Indeed, theoretical analyses in the 1990s [38, 55] predicted that any advantages in terms of asymptotic local rate of convergence will be lost in the stochastic setting with authors eventually discouraging the use of momentum [23] for stochastic gradients. In 2013, however, Sutskever et al. [51] again sparked interest in the use of momentum in the noisy setting as they argued that, while $\beta > 0$ does not help in the fine local convergence regime, it accelerates the initial phase of convergence. We have seen on the example of the effective learning rate in section 2.1.1 an instance where the literature has established the existence of an interdependence between the learning rate α and the momentum coefficient β. In fact, Sutskever et al. [51] implicitly acknowledge such an interdependence as their experiments employ tuned settings of (α, β). Given the literature's observation [20, 58, 9] that a sensible choice of the learning rate α is essential for SGD to achieve reasonable training and test performance, a natural research objective is to understand the empirical effect of a positive momentum coefficient $\beta > 0$ on the learning rate α in SGD.

Chapter 3

Contribution

As discussed in chapter 1, we make use of mathematical optimization to *train* machine learning models with the goal of obtaining predictors which *generalize* well. Our contributions can be categorized as an empirical investigation of the interdependence of the learning rate α and the momentum coefficient β in SHB and SNAG along the dimensions of optimization and generalization, concluding with a promising heuristic for large-batch training in deep learning, the *Momentum Linear Scaling Rule (MLSR)*. As both SHB and SNAG exhibit comparable results, we choose to focus our discussion in this chapter on the significantly more popular [9] SHB method to which we will refer as SGDM, i.e., "SGD with momentum", in the remainder of this work. An extension of our analysis to SNAG along with a detailed elaboration on the experimental setup can be found in Appendix .2.

The main contributions of this thesis are as follows.

- *Optimization:* Through experiments, we establish in section 3.1 that a momentum coefficient $\beta \in (0, 1)$ enlarges the learning rate of SGD.
  - For the convex setting, we conjecture in section 3.1.1 that SGD with momentum empirically has no advantage over plain SGD, thereby supporting results from convergence analysis [47, 26] in that regard.
  - For the non-convex deep learning setting, we conjecture in section 3.1.2 that, rather than being able to boost the training performance [51] per se, SGD with momentum is able to enlarge the learning rate of SGD in a way robust to unfavorable initialization points.
- *Generalization:* Through experiments, we establish in section 3.2 that a momentum coefficient $\beta \in (0, 1)$ enlarges the learning rate of SGD and thus increases the efficiency of SGD to escape from sharp local minima in the non-convex deep learning setting.
- *Momentum Linear Scaling Rule (MLSR):* Based on our findings in section 3.1.2 and section 3.2, we derive the MLSR heuristic in section 3.3 to robustly move from a small-batch setting to a large-batch setting in deep learning. Employing the MLSR for such transitions has the benefit of maintaining a sensible level of stochastic gradient noise — which is essential for good generalization — without the need for warmup strategies with respect to the learning rate.

3.1 Optimization

In the stochastic setting, the literature [62, 30, 47, 26, 61] could not identify — by means of local convergence analysis — any theoretical advantages of SGDM over plain SGD. In that regard, the usage of a positive momentum coefficient $\beta > 0$ in SGD has been in fact declared [38, 55, 23] redundant 30 years ago. In 2013, however, Sutskever et al. [51] claimed that SGDM's benefit lies in accelerating the initial phase of convergence, maintaining the same performance in the fine local convergence regime as SGD. They supported their argument with experiments, using tuned settings of (α, β). Following up on this, the literature [49, 59] has argued that the effect of $\beta > 0$ is rather a change in the effective learning rate of SGD which can be modelled [9, 59, 27] by $\tilde{\alpha} = \frac{\alpha}{1-\beta}$ as introduced in section 2.1.1, reflecting the idea that $\beta > 0$ enlarges [49] the learning rate α. Given that a sensible choice of the learning rate α is essential [20, 58, 9] for SGD to achieve reasonable training performance, we would like to examine the effect of $\beta > 0$ on a learning rate α with respect to SGDM's efficiency to minimize an objective.

Thus, this section empirically investigates the interdependence between the learning rate α and the momentum coefficient β in SGDM with regard to the algorithm's optimization performance. In that regard, section 3.1.1 examines the convex case and section 3.1.2 explores the non-convex deep learning setting.

3.1.1 Convex problems

In terms of convergence analysis, the literature could merely identify that SGDM [47] converges *as fast as* SGD under strong convexity and convexity for specific iteration-dependent schedules of (α, β). Moreover, for any constant $\beta \in (0, 1)$ — a setting which is said to empirically accelerate the training performance [9, 51] with the best practice of a fixed $\beta = 0.9$ [45, 59] — it has been recently proven [26] that there exists a Lipschitz and convex function for which the last iterate of SGDM suffers from a *suboptimal* convergence rate. That is, from the point of view of convergence analysis [47, 26, 61], the usage of $\beta > 0$ for (strongly) convex objectives, such as (L_2-regularized) logistic regression, does not provide any advantages over plain SGD in terms of boosting the training performance. Addressing this argument, Sutskever et al. [51] claimed in 2013 that, even though $\beta > 0$ has no benefit within the fine local convergence regime, SGDM is however able to accelerate the initial phase of convergence. As their claims are based on tuned settings of (α, β) with the literature suggesting that $\beta > 0$ merely enlarges [49, 59] the learning rate, we aim to empirically investigate the interdependence between the learning rate α and the momentum coefficient β in SGDM for the case of convex problems. We do so in this section by means of grid experiments of which the setup is explained below

Grid experiment: We fix a level of stochasticity in the gradients σ^2 (for example via the batch size $B \propto 1/\sigma^2$), a maximum number of iterations $T_{\max}$ and a decreasing sequence of tolerance levels $\mathcal{E} = \varepsilon_{\max}, ..., \varepsilon_{\min}$. We then consider two sets G_α and G_β which respectively include values for the learning rate α and the momentum coefficient β, where we ensure $0 \in G_\beta$ to account for the case of SGD. We then run SGDM with all $(\alpha, \beta) \in G_\alpha \times G_\beta$ on an objective function and measure the number of iterations T for the algorithm to converge to every $\varepsilon \in \mathcal{E}$ for each setting (α, β). Note that we can also describe the iteration budget through the number of epochs E, where it takes $\lceil n/B \rceil$ iterations to perform one epoch.

Experiment 1: A strongly convex objective

We would like to empirically investigate the interdependence between the learning rate α and the momentum coefficient β of SGDM with respect to the algorithm's optimization performance on a strongly convex objective by means of a grid experiment. To be able to control the level of stochasticity in the gradients directly through a parameter σ^2, we consider, the μ-strongly convex quadratic function [50] $f : \mathbb{R}^p \to \mathbb{R}$,

$$f(\boldsymbol{x}) \coloneqq \frac{1}{2}\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x}\rangle + \frac{\mu}{2}\|\boldsymbol{x}\|^2,$$

for $p = 20$, $\mu = 0.2$. $\boldsymbol{A} \in \mathbb{R}^{p\times p}$ is a band-diagonal matrix with $[-\mathbf{1}_{p-1}, 2\mathbf{1}_p, -\mathbf{1}_{p-1}]$ on the diagonals. We define stochastic gradients with $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \cdot \boldsymbol{I}_p)$ as

$$g(\boldsymbol{x}) \coloneqq \nabla f(\boldsymbol{x}) + \boldsymbol{\eta}$$

which implies that $\mathbb{E}[\|g(\boldsymbol{x}) - \nabla f(\boldsymbol{x})\|^2] \leq \sigma^2$ for all $\boldsymbol{x} \in \mathbb{R}^p$. Thus, we are able to adapt the parameter σ^2 of assumption 3 directly.

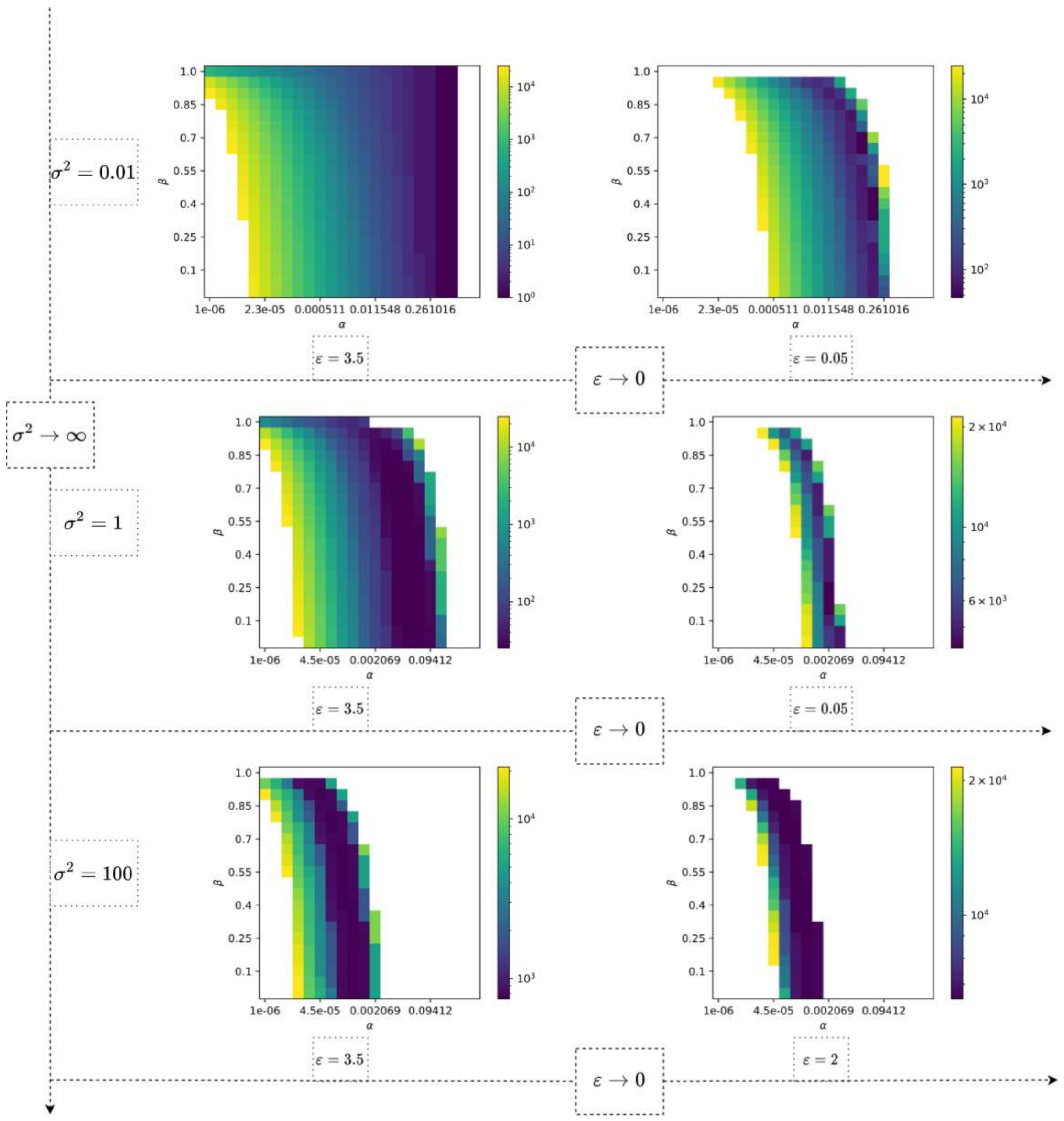


Figure 3.1: ***Experiment 1*** *[grid experiment, str. convex quadratic, median of 3 runs]:* Informally, we observe in each grid a non-white region $\mathcal{R} \subset G_\alpha \times G_\beta$ for which certain settings $(\alpha, \beta) \in G_\alpha \times G_\beta$ converged within the iteration budget. Within $\mathcal{R}$, we see color-encoded efficiency regions $\mathcal{S}_i \subset \mathcal{R}, i = 1, \ldots$ with $S_i \cap S_j = \emptyset, i \neq j$. In each $\mathcal{S}_i$, every setting $(\alpha, \beta) \in \mathcal{S}_i$ has approximately the same efficiency to converge. For instance, the dark blue pixels correspond to an efficiency region with settings (α, β) that approximately converged the fastest within $\mathcal{R}$. Within every $\mathcal{S}_i$, we observe a tendency for the highest momentum coefficient β to be associated to a setting comprising the smallest learning rate α and vice-versa. Furthermore, we observe a tendency that there always exists an $\alpha \in G_\alpha$ for which $(\alpha, 0) \in \mathcal{S}_i$ for every $i = 1, \ldots$, i.e., we see a tendency that plain SGD is able to be represented in each efficiency region. Hence, as per this experiment, we observe the phenomenon [49, 59] that SGDM with $(\alpha, \beta), \beta > 0$ tends to enlarge the learning rate of SGD.

Experiment 2: A convex objective

In grid experiment 1, we considered an objective function where we could directly manipulate the stochasticity in the gradients with a parameter. We would now like to examine the interdependence of α and β of SGDM with respect to the algorithm's training performance on a real, convex machine learning task. For that purpose, we would like to minimize the binary cross-entropy training loss of a simple logistic regression model trained on the full diabetes dataset [7]. To control the level of stochasticity, we adapt the batch size B, where the gradient estimates become noisier the smaller the batch size.

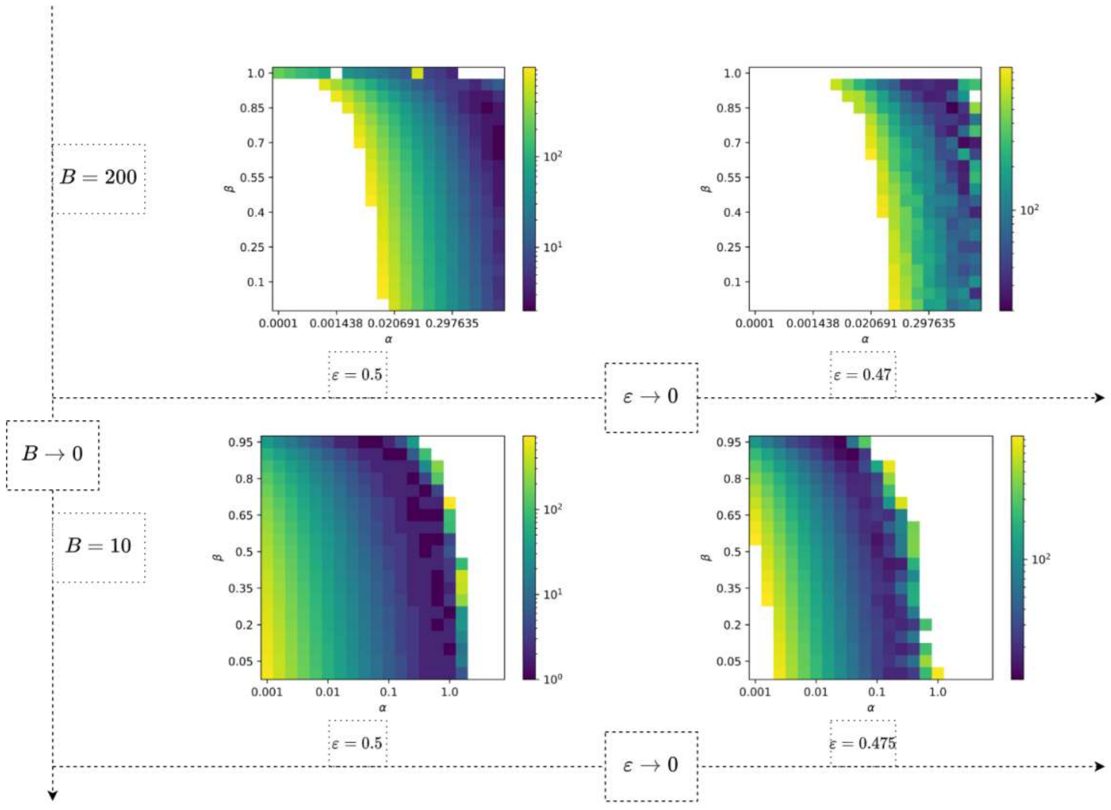


Figure 3.2: ***Experiment 2** [grid experiment; logistic regression/diabetes data; median of 3 runs]:* We immediately observe that our conclusions from the strongly convex case in experiment 1 extend to the convex case. That is, we see a tendency of SGDM to enlarge the learning rate of plain SGD.

A conclusion for the convex case

In order to empirically investigate the interdependence between the learning rate α and the momentum coefficient β with respect to the training performance of SGDM in the convex setting, we employed a grid experiment for a strongly convex objective in experiment 1 as well as a grid experiment for a convex objective in experiment 2. In both experiments, we observed that $\beta > 0$ has a tendency to enlarge the learning rate of SGD which supports similar claims in the literature [49, 59]. That is, a smaller learning rate α_s extended with $\beta > 0$ is able to reduce the training loss as efficiently as SGD with a learning rate $\alpha', \alpha' \gtrapprox \alpha_s$. Moreover, we observed that a medium learning α_m extended with $\beta > 0$ is prone to exhibit diverging behavior akin to SGD with a learning rate $\alpha'', \alpha'' \gtrapprox \alpha_m$. This observation supports the notion of an effective learning rate as introduced in section 2.1.1, reflecting the tendency that SGDM with (α, β) enlarges α in a fashion which can be modelled by $\tilde{\alpha} = \frac{\alpha}{1-\beta}$. Furthermore, observing the trajectory of the (α, β)-grids over a decreasing sequence $\mathcal{E}$ gave us insight into the convergence efficiency for a setting (α, β), under a certain level of stochasticity in the gradients, from the initial phase of convergence until fine local convergence set in. In fact, even for settings with a very low level of stochasticity in the gradient estimates, we could not observe a superiority of employing $\beta > 0$ in the initial phase of convergence as we were always able to find a setting $(\alpha, 0)$, i.e. a learning rate α for plain SGD, which was among the settings with the highest convergence efficiency. That is, we are not able to verify the claim by Sutskever et al. [51] that momentum accelerates the convergence during the initial phase in the convex case.

From a practical perspective, we can deduce from experiments 1 & 2 that we are required to decrease α over time to efficiently reach the vicinity of a minimum. This is supported by convergence analysis as introduced in section 1.3.3. Moreover, as per our experiments, we can always find a learning rate for SGD which is competitive with a specific setting of SGDM. Hence, we conjecture that a more natural approach to efficiently train a convex machine learning predictor is to fine-tune a learning rate schedule for plain SGD without the use of momentum, and by that, reduce the hyperparameter search space of the optimizer.

3.1.2 Non-convex deep learning problems

In the previous section, we examined the interdependence of α and β in SGDM with respect to SGDM's performance in training a convex machine learning predictor. We concluded from our experiments that the use of momentum in SGD for a convex problem has no advantages over plain SGD with respect to training efficiency, supporting recent theoretical observations [26, 47]. In practice, however, $\beta > 0$ in SGD is mainly [22, 51, 26] employed when training non-convex deep neural networks, where practitioners often refer to best practices [45, 9] such as $\beta = 0.9$. As for the convex case, the advantage of SGDM over plain SGD could not be captured [62, 61] with the tools of convergence analysis. Following up on this, Sutskever et al. [51] defended the use of momentum in the stochastic setting as they argued that, while momentum has no advantage in the fine local convergence regime, $\beta > 0$ helps in speeding up the initial phase of convergence. This stadium of convergence, according to their argument, is particularly important when training non-convex DNNs. In the context of local convergence analysis, said initial phase can be described by the optimization term as introduced in section 1.3.3. Indeed, for smooth non-convex objectives under assumptions 1-3, SGDM's state-of-the-art convergence result [62] for a fixed α,

$$\mathcal{O}\left(\frac{1}{\frac{\alpha}{1-\beta}T} + \frac{\alpha}{1-\beta}\sigma^2\right),$$

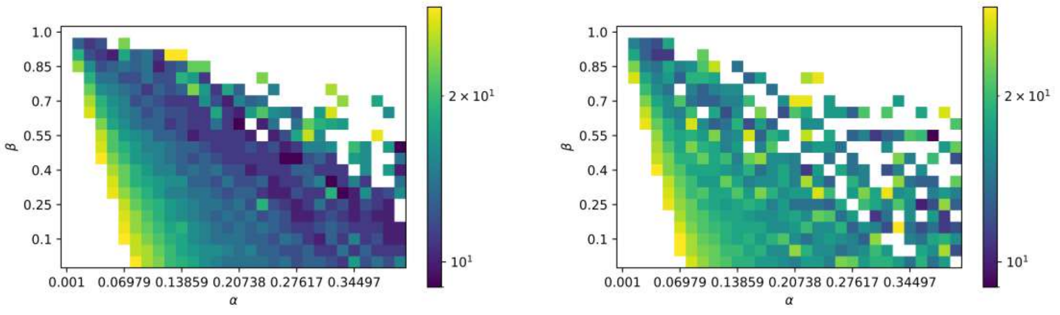
corresponds to the rate achieved by plain SGD with a learning rate of $\frac{\alpha}{1-\beta}$. That is, we again notice an indication that SGDM with (α, β) enlarges [49, 59] the learning rate of SGD, behaving similarly to SGD with an effective learning rate of $\tilde{\alpha}$.

Given that a sensible choice of the learning rate α is essential [20, 58, 9] for SGD to achieve reasonable training performance in deep learning problems, we would like to examine the effect of $\beta > 0$ on a learning rate α with respect to SGDM's efficiency to converge to the vicinity of a local minimum. In particular, we want to take up on the indication [62, 59, 27] that SGDM with (α, β) performs similarly to SGD with $\tilde{\alpha}$.

Beginning our analysis, we will observe in experiment 3 the effect of starting SGDM at different initialization points $\boldsymbol{x}_1$ within the non-convex and non-linear landscape of a DNN by means of a grid experiment. In experiment 4, we will follow up on the findings of experiment 3, investigating the behavior of SGDM with (α, β) and SGD with $\tilde{\alpha}$ for increasing learning rates when started from different initialization points $\boldsymbol{x}_1$.

Experiment 3: A non-convex deep learning objective

In section 3.1.1, we introduced the setup of a grid experiment to empirically investigate the interdependence of α and β in SGDM with respect to the method's training efficiency. In this experiment, we conduct two grid experiments for a deep learning problem, using two different initialization points $\boldsymbol{x}_1$ for SGDM. In particular, we will minimize the cross-entropy training loss of the simple neural network LeNet [24] trained on the Fashion-MNIST [56] dataset.



(a) initialization 1. (b) initialization 2.

Figure 3.3: ***Experiment 3*** *[grid experiment; LeNet/FMNIST; 2 different init. points]:* We observe the result of a grid experiment for two different initialization points and notice that we are confronted with two quite varying grids due to the non-convexity and non-linearity of the objective. In particular, we observe that initialization 1. in figure 3a induces a grid exhibiting geometrically similar color-encoded efficiency regions as the (strongly) convex case, while initialization 2. in figure 3b induces a grid which exhibits a region $\mathcal{R} \subset G_\alpha \times G_\beta$ which is comparatively less connected above its "diagonal", i.e., the region within the grid which has α and β relatively large comprises settings which could converge within the epoch budget with neighboring settings that could not. In particular, we notice that settings (α, β) which have α relatively small and β relatively large are able to converge for both initialization points while being represented in relatively high efficiency regions.

Experiment 4: The robustness of the effective learning rate

In experiment 3, we observed that, for one initialization point, SGDM has a tendency to enlarge the learning rate of SGD in a fashion that aligns with our conclusion for the convex case in section 3.1.1 and which supports the literature's indication [62, 59, 49, 27] that SGDM with (α, β) behaves similarly to SGD with $\tilde{\alpha}$, providing no tangible advantage over plain SGD. However, for another initialization point, we observe a tendency for plain SGD with a larger learning rate not to be competitive with SGDM. Hence, experiment 3 could be seen as an indication that, only for a favorable initialization point, SGD with a larger learning rate behaves similarly to SGDM. By extension, this observation could suggest that the training performance of SGDM with (α, β) is more robust to where we start the method within the non-convex landscape than SGD with $\tilde{\alpha}$. To follow up on this observation, we would like to empirically investigate the training loss achieved by SGDM with (α, β) and SGD with $\tilde{\alpha}$ over numerous initialization points. To that end, we will again minimize the cross-entropy training loss of the neural network LeNet [24] trained on the Fashion-MNIST [56] dataset.

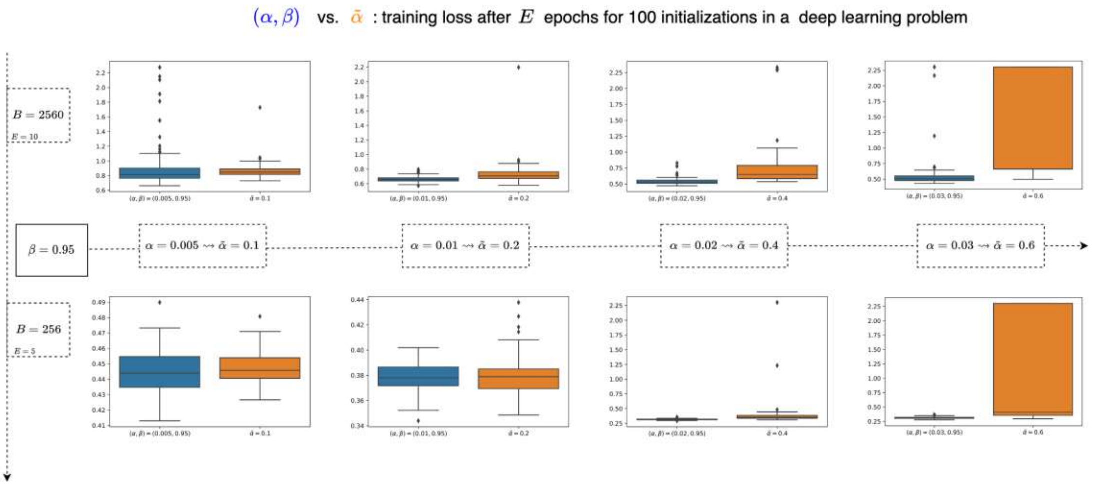


Figure 3.4: ***Experiment 4*** *[(α, β) vs. $\tilde{\alpha}$ for $\tilde{\alpha} \to \infty$; LeNet/FMNIST; 100 init. points; training loss after E epochs]:* For a fixed $\beta = 0.95$, we observe that SGD (blue) with $\tilde{\alpha}$ small is able to achieve a comparable training loss distribution after E epochs to SGDM (orange) with (α, β) over 100 initializations. However, as we increase $\tilde{\alpha}$, SGD has a tendency to be more prone to indulge in bad convergence behavior than SGDM with (α, β), while being able to approximately achieve the competitor's minimum training loss for favorable initializations. We can hence conjecture that SGDM with (α, β) tends to be more robust to initialization points than SGD with $\tilde{\alpha}$ when $\tilde{\alpha}$ becomes larger. Note that this behavior appears to be invariant to a positive scaling of the batch size B.

A conclusion for the non-convex deep learning case

We deduce from experiments 3 & 4 that, for small learning rates α, SGDM with (α, β) exhibits a training performance similar to SGD with a larger learning rate, supporting the literature's [62, 59, 49, 57] indication that SGDM induces a learning rate $\tilde{\alpha}$ for plain SGD. For larger learning rates α, however, SGDM with (α, β) tends to converge as efficiently as SGD with $\tilde{\alpha}$ initialized at a favorable initialization point, i.e., a starting point $\boldsymbol{x}_1$ from which SGD is able to converge efficiently [20] with relatively large learning rates. Synthesizing our findings, we can conjecture that SGDM is able to perform larger effective steps[1] than SGD for a multitude of initialization points. That is, rather than being able to increase the training speed [51] in the initial phase of convergence per se — a claim which could not be supported by convergence analysis [62, 61] —, SGDM is able to boost the training performance conditional to an unfavorable initialization point $\boldsymbol{x}_1$ for plain SGD.

In practical terms, we conclude that employing $\beta > 0$ represents an alternative to increasing the learning rate α of SGD robust to the initialization point. That is, the best practice of using a momentum coefficient $\beta = 0.9$ [45] with a smaller learning rate helps SGD reduce the training loss more reliably than employing a larger learning rate.

[1]We note that table 2 of [27] extends our observations to ResNet-9 [12] trained on CIFAR-100 [21].

3.2 Generalization

We have discussed in chapter 1 that our goal when training a machine learning predictor is to obtain a model which does not overfit the training set in order to generalize well to unseen data.

For convex problems, we can mitigate overfitting by adapting the objective itself using regularization. Hence, in the convex case, generalization efficiency aligns directly with optimization efficiency as our aim is to reduce the empirical risk of the regularized objective. Thus, as per our results from section 3.1.1, SGDM provides neither theoretical [47, 26] nor empirical advantages over SGD to reduce the regularized training loss of a convex machine learning predictor and, by extension, does not benefit generalization in the convex case.

For non-convex deep learning objectives, generalization efficiency aligns with both optimization efficiency and the efficiency to escape from sharp local minima as discussed in section 1.4. Indeed, to achieve good generalization within a time budget for a deep learning problem, we are required to converge fast to a flat local minimum. Hence, when training a DNN with SGD, we need [57] the stochastic gradient noise [32] $\frac{\alpha}{B}$ to be sufficiently large in order to escape from sharp minima during our quest for a flat minimum while being constrained [10, 34, 57] with respect to the range of α in order to achieve good convergence behavior.

In this section, we would like to to empirically investigate the interdependence of α and β in SGDM with respect to the algorithm's efficiency to escape from sharp minima, extending our analysis of section 3.1.2 to the second important pillar of generalization performance in deep learning.

Experiment 5: Escaping sharp valleys

In this experiment, we would like to empirically investigate the interdependence of the learning rate α and the momentum coefficient β in SGDM with respect to the algorithm's efficiency to escape from sharp minima, considering different levels of sharpness under varying levels of noise in the gradients. To that end, we employ a setup analogous to the grid experiments introduced in section 3.1.1, where we report the number of iterations for the setting (α, β) to escape from a minimum. Inspired by [64, 50], we consider the non-convex Styblinski-Tang function $f : \mathbb{R}^p \to \mathbb{R}$,

$$f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{p}(x_i^4 - 16x_i^2 + 5x_i),$$

for $p = 10$ and define stochastic gradients with $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \cdot \boldsymbol{I}_p)$ as

$$g(\boldsymbol{x}) \coloneqq \nabla f(\boldsymbol{x}) + \boldsymbol{\eta}$$

which implies that $\mathbb{E}[\|g(\boldsymbol{x}) - \nabla f(\boldsymbol{x})\|^2] \leq \sigma^2$ for all $\boldsymbol{x} \in \mathbb{R}^p$. As we multiply every parameter of f by $\sqrt{k}$, we rescale the Hessian of f by a factor of k since $f(\sqrt{k}\boldsymbol{x}) \rightsquigarrow k\nabla^2 f(\boldsymbol{x})$ for a point $\boldsymbol{x} \in \mathbb{R}^p$. Thus, we are able to control the stochasticity in the gradients by a parameter σ^2 and the relative minima sharpness by a parameter k.

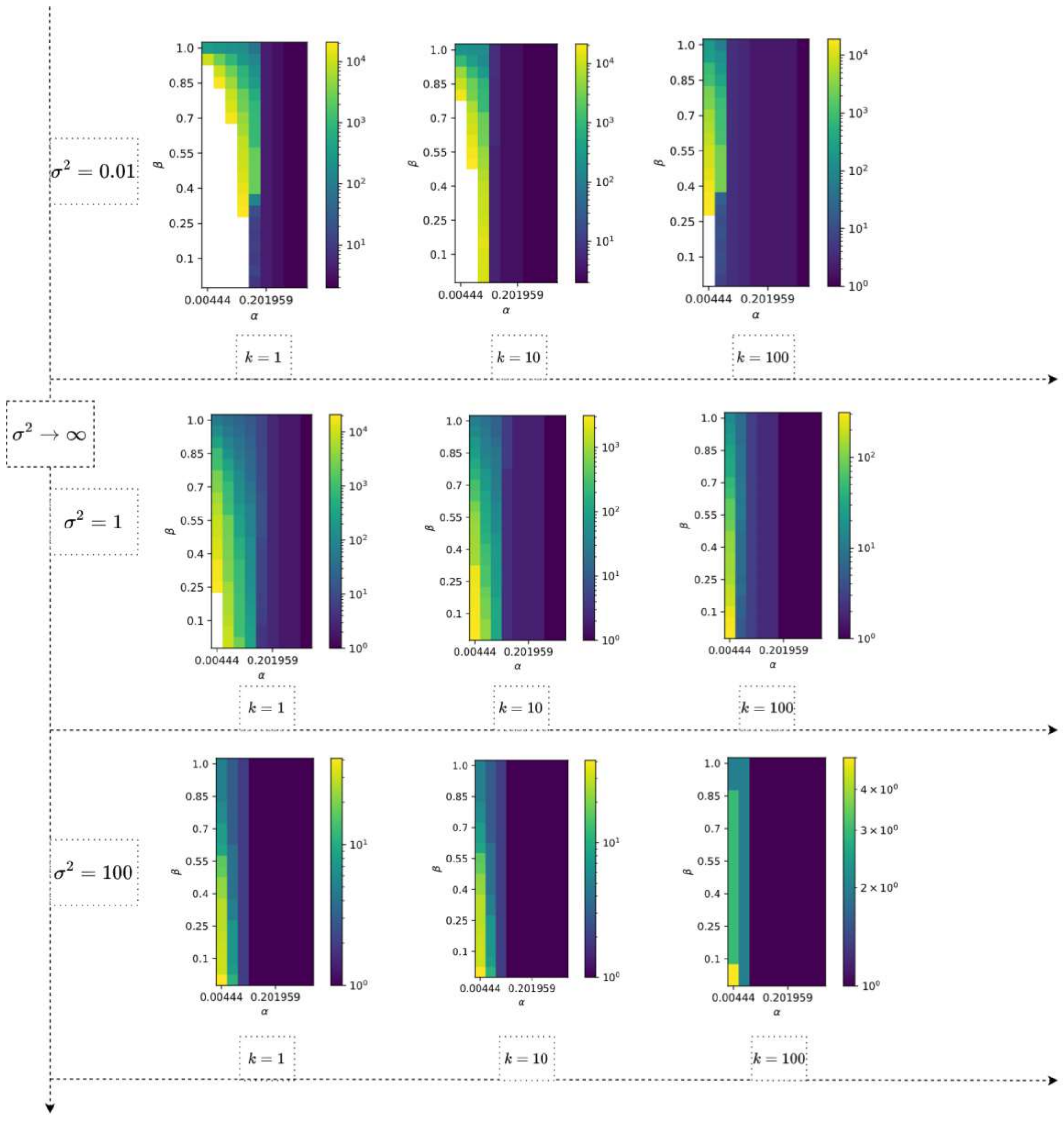


Figure 3.5: ***Experiment 5*** *[sharpness grid experiment; Styblinski-Tang function; median of 3 runs]:* Tying in with our conclusions of section 3.1.2, we can observe that SGDM has a tendency to enlarge the learning rate of plain SGD with respect to the method's efficiency to escape from sharp minima. Moreover, we observe that $\sigma^2 \to \infty$ increases the efficiency to escape from a minimum for smaller learning rates, thereby confirming theoretical results [32].

Experiment 6: Reducing sharpness with large batches

In experiment 5, we considered a non-convex objective function where we could directly manipulate the sharpness of a minimum with a parameter. We now would like to examine the interdependence of α and β in SGDM with respect to the algorithm's efficiency in avoiding sharp regions on a real non-convex deep learning task. To that end, we train the neural network LeNet [24] on a subsample of the Fashion-MNIST dataset [56] and find a local minimum $\bar{\boldsymbol{x}}_{GD}$ with GD, achieving a cross-entropy training loss of ≈ 0. As discussed in section 1.4, local minima found by $B = n$ are usually relatively sharp and associated with poor [28, 64] generalization. We then employ a setup analogous to the grid experiments introduced in section 3.1.1, where we report the number of epochs for a large-batch setting of (α, β) initialized at $\bar{\boldsymbol{x}}_{GD}$ to reduce a sharpness estimate to a target value. Although we defined sharpness in section 1.4 based on the eigenvalues of the Hessian at the corresponding local minimum, it is usually too expensive to compute $\nabla^2 f(\cdot)$ in practice. Hence, we employ the following estimate [64] of the sharpness at a point $\boldsymbol{x} \in \mathbb{R}^p$ within a non-convex deep learning objective f,

$$\frac{1}{M}\sum_{i=1}^{M} f(\boldsymbol{x} + \boldsymbol{\eta}_i) - f(\boldsymbol{x}), \quad \boldsymbol{\eta}_i \sim \mathcal{N}(0, \delta^2 \cdot \boldsymbol{I}_p), \tag{3.1}$$

where $M = 50$ and $\delta = 0.01$.

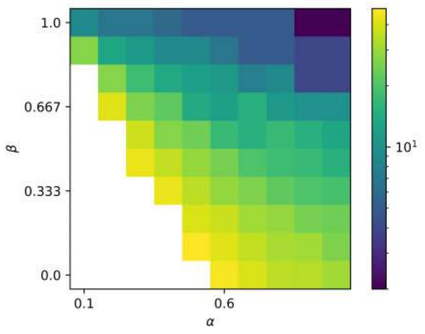


Figure 3.6: ***Experiment 6*** *[large-batch sharpness grid experiment; LeNet/subsampled FMNIST]:* We observe that the color-encoded efficiency regions follow a diagonal pattern within the grid, indicating that $\beta > 0$ enlarges the learning rate of plain SGD.

A conclusion for generalization in deep learning.

We have seen in experiments 5 & 6 that a momentum coefficient $\beta \in (0, 1)$ has a tendency to enlarge the learning rate of plain SGD with respect to the method's efficiency to escape from sharp minima. This finding supports the argument [27, 9, 64, 59] that SGDM with (α, β) can be approximated by SGD with $\tilde{\alpha} = \frac{\alpha}{1-\beta}$.

3.3 The Momentum Linear Scaling Rule (MLSR)

Given our results from sections 3.1.2 and 3.2, we can conjecture that SGDM with (α, β) is more likely to exhibit good generalization efficiency as it is better equipped than SGD to achieve good convergence performance while keeping a sensible amount of stochastic gradient noise (SGN) to escape from sharp minima. Indeed, the literature argues [32] that the SGN of SGDM with (α, β) is proportional to $\frac{\alpha}{B(1-\beta)}$ — reflecting again an effective learning rate of $\tilde{\alpha} = \frac{\alpha}{1-\beta}$ for SGD. While it is claimed [32] that numerous settings of α, β, B are able to achieve the same level of SGN, and by that, lead to comparable escaping efficiency, we have established in section 3.1.2 that SGDM tends to be more robust to the initialization point $\boldsymbol{x}_1$ and, by extension, is more likely to achieve good convergence behavior.

As discussed in section 1.4.1, large-batch training is able to efficiently utilize parallel computation to accelerate the training of deep neural networks. In order to maintain a similar magnitude of SGN, the literature has advocated [10] to multiply the learning rate by c when the batch size is multiplied by $c > 1$ — a heuristic called the *Linear Scaling Rule (LSR)*. However, as an increase of the learning rate α only works in a narrow [10, 34, 57] range, since too large initial learning rates may lead to optimization divergence or bad convergence, this rule has to be applied with care, with the authors proposing [10] specific warmup strategies for α. Given our conclusions from sections 3.1.2 and 3.2, we present an extension[2] of this heuristic which naturally overcomes optimization challenges early in training through the use of momentum.

Momentum Linear Scaling Rule (MLSR):

For a setting (α, β, B), when the batch size B is multiplied by $c > 1$, adapt the momentum coefficient as

$$\beta := 1 - \frac{1-\beta}{c}.$$

For plain SGD ($\beta = 0$), we thus introduce a momentum coefficient $\beta = 1 - 1/c$.

[2]This rule is derived from our goal of maintaining the same level of SGN for a new batch size $cB, c > 1$ by adapting the momentum coefficient. That is, we set $\frac{\alpha}{cB(1-\beta_c)} = \frac{\alpha}{B(1-\beta)}$ and solve for β_c.

In experiments 8 and 9 we will illustrate the benefit of the MLSR in practice. In particular, we will show akin to the original LSR [10] paper that, while keeping all hyperparameters — including the number of epochs — unchanged, we are able to closely match the top-1 test error of a small-batch benchmark, the "Base" case, without the need for extensive warmup strategies for α. To that end, we train the neural networks VGG11 [48] and ResNet-18 [12] on the CIFAR-10 [21] dataset.

(α, β, B) $\boxed{\text{Base:}(0.1, 0, 100)}$ $\boxed{\text{LSR:}(2, 0, 2000)}$ $\boxed{\text{MLSR:}(0.1, 0.95, 2000)}$

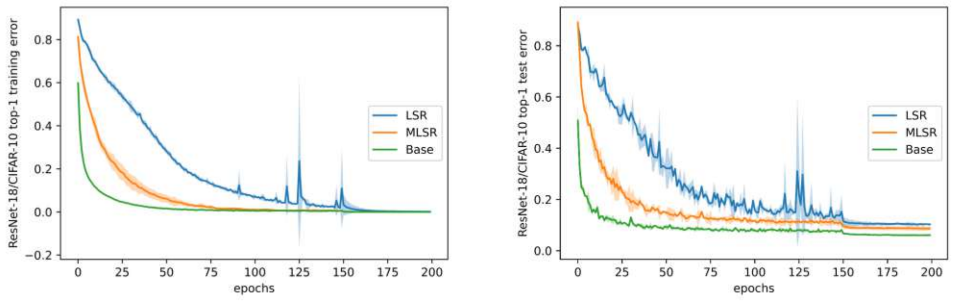


Figure 3.7: ***Experiment 7*** *[MLSR; ResNet-18/CIFAR-10; c=20; 2 runs; error range of* $\pm 2 \times$ *std]:* We observe that, with the MLSR, we are able to closely match the top-1 training and test error curves of a small-batch benchmark when we scale the batch size by 20. For the LSR, however, we observe comparatively non-monotonic test performance.

(α, β, B) $\boxed{\text{Base:}(0.05, 0, 100)}$ $\boxed{\text{LSR:}(1, 0, 2000)}$ $\boxed{\text{MLSR:}(0.05, 0.95, 2000)}$

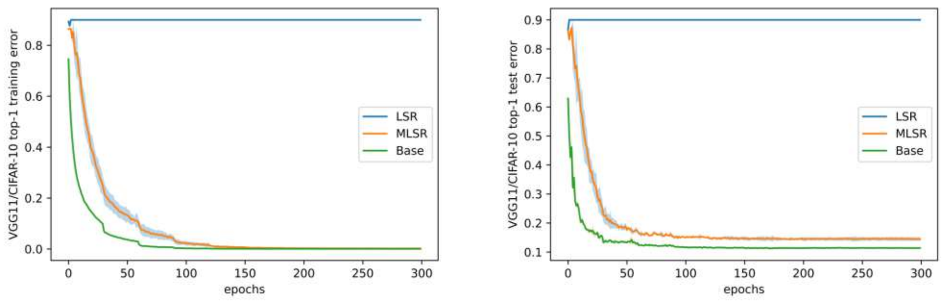


Figure 3.8: ***Experiment 8*** *[MLSR; VGG11/CIFAR-10; c=20; 2 runs; error range of* $\pm 2 \times$ *std]:* We observe that, with the MLSR, we are able to closely match the top-1 training and test error curves of a small-batch benchmark when we scale the batch size by 20. With the LSR, however, we would diverge without the use of warmup strategies for the learning rate.

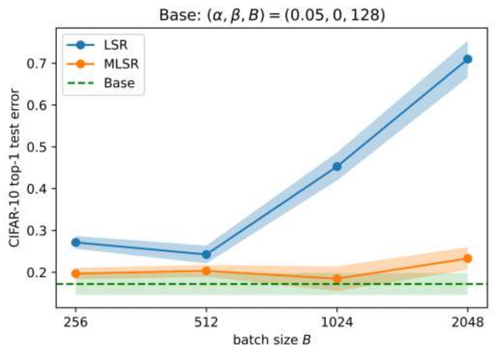


Figure 3.9: ***Experiment 9*** *[MLSR mult. batch sizes; test error after 10 epochs; ResNet-18/CIFAR-10; 3 runs; error range of* $\pm 2\times$*std]:* We see that the MLSR is able to approximately maintain the top-1 test error after 10 epochs of the base case for a batch size scaling of up to 16 while the LSR would induce bad convergence behavior without the employment of special warmup strategies.

Chapter 4

Conclusion

In the deterministic case, GD extended with the notion of momentum has a *provable* [8] advantage over plain GD. In spite of that, we conjectured in chapter 3 that, based on our empirical results, a momentum coefficient $\beta \in (0, 1)$ in the stochastic setting generally has the effect of merely enlarging the learning rate of plain SGD, not contributing to a boost in performance per se. That is, for the convex case, we supported in section 3.1.1 the results from convergence analysis [26, 47] as we could not empirically establish an advantage of SGD with momentum over plain SGD. For the non-convex deep learning case, we concluded in section 3.1.2 that SGD with momentum empirically has the effect of enlarging the learning rate in a more robust fashion. In particular, rather than being capable of faster convergence [51] per se — a claim which could not be supported by convergence analysis [62, 61] —, SGD with momentum is able to improve the training performance conditional to an unfavorable initialization point for plain SGD. In addition to that, we empirically established in section 3.2 that $\beta > 0$ has the ability to compensate for small learning rates with respect to the method's efficiency to escape from sharp minima. Based on our findings in sections 3.1.2 and 3.2, we were able to derive a heuristic, the MLSR, for transitioning from small-batch to large-batch training in deep learning while approximately maintaining the same generalization efficiency. We believe that it would be of great practical interest to benchmark and test the MLSR heuristic against existing rules [10], accounting for a variety [34] of warmup strategies, complex deep learning models and large datasets such as ImageNet [6].

Bibliography

[1] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. *Optimization Methods for Large-Scale Machine Learning*. 2018. arXiv: `1606.04838`. URL: `https://arxiv.org/abs/1606.04838`.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). arXiv: `2005.14165`. URL: `https://arxiv.org/abs/2005.14165`.

[3] A. Cauchy. "Methode generale pour la resolution des systemes d'equations simultanees". In: *C.R. Acad. Sci. Paris* 25 (1847), pp. 536–538. URL: `https://ci.nii.ac.jp/naid/10026863174/en/`.

[4] Ashok Cutkosky and Harsh Mehta. "Momentum Improves Normalized SGD". In: *CoRR* abs/2002.03305 (2020). arXiv: `2002.03305`. URL: `https://arxiv.org/abs/2002.03305`.

[5] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[7] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[8] Euhanna Ghadimi, Hamid Reza Feyzmahdavian, and Mikael Johansson. "Global convergence of the Heavy-ball method for convex optimization". In: *2015 European Control Conference (ECC)* (2015), pp. 310–315.

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

[10] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". In: *CoRR* abs/1706.02677 (2017). arXiv: `1706.02677`. URL: `http://arxiv.org/abs/1706.02677`.

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[13] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(91)90009-T`. URL: `https://www.sciencedirect.com/science/article/pii/089360809190009T`.

[14] Prateek Jain, Sham M. Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. "Accelerating Stochastic Gradient Descent For Least Squares Regression". In: *arXiv e-prints* (Apr. 2017). arXiv: `1704.08227`. URL: `https://arxiv.org/abs/1704.08227`.

[15] Prateek Jain and Purushottam Kar. "Non-convex Optimization for Machine Learning". In: *Foundations and Trends® in Machine Learning* 10.3-4 (2017), pp. 142–336. DOI: `10.1561/2200000058`. URL: `https://doi.org/10.1561%2F2200000058`.

[16] Michael Jordan. *Why the logistic function? A tutorial discussion on probabilities and neural networks*. Tech. rep. Massachusetts Institute of Technology, 1995.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836 (2016). arXiv: `1609.04836`. URL: `http://arxiv.org/abs/1609.04836`.

[18] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. "On the insufficiency of existing momentum schemes for Stochastic Optimization". In: *CoRR* abs/1803.05591 (2018). arXiv: `1803.05591`. URL: `http://arxiv.org/abs/1803.05591`.

[19] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014), arXiv:1412.6980. arXiv: `1412.6980`. URL: `https://arxiv.org/abs/1412.6980.pdf`.

[20] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. "An Alternative View: When Does SGD Escape Local Minima?" In: *CoRR* abs/1802.06175 (2018). arXiv: `1802.06175`. URL: `http://arxiv.org/abs/1802.06175`.

[21] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[23] Yann Lecun, Leon Bottou, Genevieve Orr, and Klaus-Robert Müller. "Efficient BackProp". In: (Aug. 2000).

[24] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[25] Kfir Y. Levy, Ali Kavis, and Volkan Cevher. "STORM+: Fully Adaptive SGD with Momentum for Nonconvex Optimization". In: *arXiv e-prints* (Nov. 2021). arXiv: `2111.01040`. URL: `https://arxiv.org/abs/2111.01040`.

[26] Xiaoyun Li, Mingrui Liu, and Francesco Orabona. "On the Last Iterate Convergence of Momentum Methods". In: *ArXiv* abs/2102.07002 (2022).

[27] Tianyi Liu, Zhehui Chen, Enlu Zhou, and Tuo Zhao. "Toward Deeper Understanding of Nonconvex Stochastic Optimization with Momentum using Diffusion Approximations". In: *CoRR* abs/1802.05155 (2018). arXiv: `1802.05155`. URL: `http://arxiv.org/abs/1802.05155`.

[28] Tianyi Liu, Zhehui Chen, Enlu Zhou, and Tuo Zhao. "Toward Deeper Understanding of Nonconvex Stochastic Optimization with Momentum using Diffusion Approximations". In: *CoRR* abs/1802.05155 (2018). arXiv: `1802.05155`. URL: `http://arxiv.org/abs/1802.05155`.

[29] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. “A survey of deep neural network architectures and their applications”. In: *Neurocomputing* 234 (2017), pp. 11–26. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2016.12.038`. URL: `https://www.sciencedirect.com/science/article/pii/S0925231216315533`.

[30] Yanli Liu, Yuan Gao, and Wotao Yin. “An Improved Analysis of Stochastic Gradient Descent with Momentum”. In: *arXiv e-prints*, arXiv:2007.07989 (July 2020), arXiv:2007.07989. arXiv: `2007.07989`. URL: `https://arxiv.org/abs/2007.07989`.

[31] Jerry Ma and Denis Yarats. *Quasi-hyperbolic momentum and Adam for deep learning*. 2018. arXiv: `1810.06801`. URL: `https://arxiv.org/abs/1810.06801`.

[32] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. “Stochastic Gradient Descent as Approximate Bayesian Inference”. In: (2017). DOI: `10.48550/ARXIV.1704.04289`. URL: `https://arxiv.org/abs/1704.04289`.

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[34] Dominic Masters and Carlo Luschi. “Revisiting Small Batch Training for Deep Neural Networks”. In: *CoRR* abs/1804.07612 (2018). arXiv: `1804.07612`. URL: `http://arxiv.org/abs/1804.07612`.

[35] Rotem Mulayoff and Tomer Michaeli. “Unique Properties of Wide Minima in Deep Networks”. In: *CoRR* abs/2002.04710 (2020). arXiv: `2002.04710`. URL: `https://arxiv.org/abs/2002.04710`.

[36] Yurii Nesterov. “A method for solving the convex programming problem with convergence rate O(1/k$\hat{2}$)”. In: *Proceedings of the USSR Academy of Sciences* 269 (1983), pp. 543–547.

[37] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. 1st ed. Springer Publishing Company, Incorporated, 2014. ISBN: 1461346916.

[38] Genevieve Beth Orr. “Dynamics and Algorithms for Stochastic Search”. UMI Order No. GAX96-08998. PhD thesis. USA, 1996.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[40] Sebastian Pokutta. *Cheat Sheet: Acceleration from First Principles*. http://www.pokutta.com/blog/research/2019/06/09/cheatsheet-acceleration-first-principles.html. Accessed: 09.06.2022. 2014.

[41] B.T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[42] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical Text-Conditional Image Generation with CLIP Latents". In: *arXiv e-prints*, arXiv:2204.06125 (Apr. 2022), arXiv:2204.06125. arXiv: 2204.06125. URL: https://arxiv.org/abs/2204.06125.

[43] Sebastian Raschka. *Naive Bayes and Text Classification - Introduction and Theory*. https://sebastianraschka.com/Articles/2014_naive_bayes_1.html. Accessed: 09.06.2022. 2014.

[44] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586. URL: https://doi.org/10.1214/aoms/1177729586.

[45] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[46] Othmane Sebbouh, Robert Gower, and Aaron Defazio. *On the convergence of the Stochastic Heavy Ball Method*. June 2020.

[47] Othmane Sebbouh, Robert M. Gower, and Aaron Defazio. "On the convergence of the Stochastic Heavy Ball Method". In: *CoRR* abs/2006.07867 (2020). arXiv: 2006.07867. URL: https://arxiv.org/abs/2006.07867.

[48] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv e-prints*, arXiv:1409.1556 (Sept. 2014), arXiv:1409.1556. arXiv: `1409.1556`. URL: `https://arxiv.org/abs/1409.1556`.

[49] Leslie N. Smith. “A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay”. In: *CoRR* abs/1803.09820 (2018). arXiv: `1803.09820`. URL: `http://arxiv.org/abs/1803.09820`.

[50] Sebastian U. Stich, Amirkeivan Mohtashami, and Martin Jaggi. “Critical Parameters for Scalable Distributed Learning with Large Batches and Asynchronous Updates”. In: *CoRR* abs/2103.02351 (2021). arXiv: `2103.02351`. URL: `https://arxiv.org/abs/2103.02351`.

[51] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. “On the importance of initialization and momentum in deep learning”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147. URL: `https://proceedings.mlr.press/v28/sutskever13.html`.

[52] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[53] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 1982. ISBN: 0387907335.

[54] Kun Wei, Yike Zhang, Sining Sun, Lei Xie, and Long Ma. “Conversational Speech Recognition By Learning Conversation-level Characteristics”. In: *arXiv e-prints*, arXiv:2202.07855 (Feb. 2022), arXiv:2202.07855. arXiv: `2202.07855`. URL: `https://arxiv.org/abs/2202.07855`.

[55] W Wiegerinck, A Komoda, and T Heskes. “Stochastic dynamics of learning with momentum in neural networks”. In: *Journal of Physics A: Mathematical and General* 27.13 (July 1994), pp. 4425–4437. DOI: `10.1088/0305-4470/27/13/017`. URL: `https://doi.org/10.1088/0305-4470/27/13/017`.

[56] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: `1708.07747`. URL: `https://arxiv.org/abs/1708.07747`.

[57] Zeke Xie, Issei Sato, and Masashi Sugiyama. *A Diffusion Theory For Deep Learning Dynamics: Stochastic Gradient Descent Exponentially Favors Flat Minima*. 2020. DOI: `10.48550/ARXIV.2002.03495`. URL: `https://arxiv.org/abs/2002.03495`.

[58] Zeke Xie, Issei Sato, and Masashi Sugiyama. "Stable Weight Decay Regularization". In: *CoRR* abs/2011.11152 (2020). arXiv: 2011.11152. URL: https://arxiv.org/abs/2011.11152.

[59] Zeke Xie, Li Yuan, Zhanxing Zhu, and Masashi Sugiyama. "Positive-Negative Momentum: Manipulating Stochastic Gradient Noise to Improve Generalization". In: *CoRR* abs/2103.17182 (2021). arXiv: 2103.17182. URL: https://arxiv.org/abs/2103.17182.

[60] Yan Yan, Tianbao Yang, Zhe Li, Qihang Lin, and Yi Yang. "A Unified Analysis of Stochastic Momentum Methods for Deep Learning". In: *arXiv e-prints*, arXiv:1808.10396 (Aug. 2018), arXiv:1808.10396. arXiv: 1808.10396. URL: https://arxiv.org/abs/1808.10396.

[61] Tianbao Yang, Qihang Lin, and Zhe Li. *Unified Convergence Analysis of Stochastic Momentum Methods for Convex and Non-convex Optimization*. 2016. arXiv: 1604.03257. URL: https://arxiv.org/abs/1604.03257.

[62] Hao Yu, Rong Jin, and Sen Yang. *On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization*. 2019. eprint: 1905.03817. URL: https://arxiv.org/abs/1905.03817.

[63] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven C. H. Hoi, and Weinan E. "Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning". In: *CoRR* abs/2010.05627 (2020). arXiv: 2010.05627. URL: https://arxiv.org/abs/2010.05627.

[64] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. *The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Sharp Minima and Regularization Effects*. 2018. DOI: 10.48550/ARXIV.1803.00195. eprint: 1803.00195. URL: https://arxiv.org/abs/1803.00195.

.1 SHB & SNAG Formulations

In this section, we briefly discuss the most important formulations for SHB and SNAG used in the literature. For instance, the papers [31, 19, 25, 14, 18, 59, 4] employ or adapt these formulations to derive related methods.

.1.1 SHB Formulations

For the iterations $t = 1, ...,$ Polyak's original formulation [41] of SHB is given as

$$\text{SHB:} \quad \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha g(\boldsymbol{x}_t) + \beta(\boldsymbol{x}_t - \boldsymbol{x}_{t-1}), \tag{1}$$

where $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and $\beta \in [0, 1)$. For $\boldsymbol{m}_0 = \boldsymbol{0}$, we can equivalently [61] express (1) as

$$\text{MOM1\_SHB:} \quad \begin{cases} \boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} - \alpha g(\boldsymbol{x}_t), \\ \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{m}_t. \end{cases} \tag{2}$$

Moreover [46], we can express (1) as

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \alpha \left[g(\boldsymbol{x}_t) - \beta \frac{(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})}{\alpha} \right] \\ &= \boldsymbol{x}_t - \alpha \left[g(\boldsymbol{x}_t) + \beta \frac{(\boldsymbol{x}_{t-1} - \boldsymbol{x}_t)}{\alpha} \right] \\ &= \boldsymbol{x}_t - \alpha \underbrace{\left[g(\boldsymbol{x}_t) + \beta \boldsymbol{m}_{t-1} \right]}_{\boldsymbol{m}_t}, \end{aligned} \tag{3}$$

since

$$\boldsymbol{m}_{t-1} = \frac{(\boldsymbol{x}_{t-1} - \boldsymbol{x}_t)}{\alpha} \implies \boldsymbol{x}_t = \boldsymbol{x}_{t-1} - \alpha \boldsymbol{m}_{t-1}, \tag{4}$$

which gives us the formulation used in PyTorch's [39] SGD optimizer when choosing a `momentum` parameter in $(0, 1)$:

$$\text{MOM2\_SHB:} \quad \begin{cases} \boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + g(\boldsymbol{x}_t), \\ \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha \boldsymbol{m}_t. \end{cases} \tag{5}$$

In [19, 31], a "normalized" version of SHB is discussed given as

$$\text{EMAM:} \quad \begin{cases} \boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + (1 - \beta) g(\boldsymbol{x}_t), \\ \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha \boldsymbol{m}_t. \end{cases} \tag{6}$$

The formulation in (6) allows us to interpret the sequence $\boldsymbol{m}_t$ as an exponential moving average of past stochastic gradients and corresponds to PyTorch's SGD optimizer with parameters `momentum=`β, `dampening=`$1-\beta, \beta \in (0,1)$. Note that EMAM has a smoothing effect [30, 63] on the noise level of the gradients while MOM2_SHB elevates [27] the variance of the stochastic gradients.

.1.2 SNAG Formulations

The original SNAG formulation by Nesterov [36] for the iterations $t = 0, \dots$ is given as

$$\text{SNAG:} \quad \begin{cases} \boldsymbol{x}_{t+1} = \boldsymbol{y}_t - \alpha g(\boldsymbol{y}_t), \\ \boldsymbol{y}_{t+1} = \boldsymbol{x}_{t+1} + \beta(\boldsymbol{y}_{t+1} - \boldsymbol{y}_t), \end{cases} \tag{7}$$

with $\boldsymbol{y}_0 = \boldsymbol{x}_0$. For $\boldsymbol{m}_t \coloneqq \boldsymbol{y}_t - \boldsymbol{y}_{t-1}$ and $\boldsymbol{m}_0 = \boldsymbol{0}$, we can relate [51] SNAG to MOM1_SHB by

$$\text{MOM\_SNAG:} \quad \begin{cases} \boldsymbol{m}_{t+1} = \beta \boldsymbol{m}_t - \alpha g(\boldsymbol{x}_t + \beta \boldsymbol{m}_t), \\ \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{m}_{t+1}, \end{cases} \tag{8}$$

which corresponds to the formulation used in PyTorch's [39] SGD optimizer.

.2 Experimental Setup

In this section, we extend our experiments to SNAG and provide additional information on the experimental setup. All experiments involving real machine learning tasks, i.e., experiments $\{2, 3, 4, 6, 7, 8, 9\}$, have been implemented using Python 3.7.13 [52], NumPy 1.21.6 [11] and PyTorch 1.11.0 [39] and run on a Tesla P100-PCIE-16GB GPU, where we used the parameters `lr,momentum, nesterov` in `torch.optim.SGD` to respectively control α, β and the choice between SHB and SNAG. The remaining experiments $\{1, 5\}$ were implemented using Python 3.7.13 and NumPy 1.21.6 and have been run on an Intel(R) Xeon(R) CPU @ 2.20GHz. The SHB and SNAG methods used in experiments $\{1, 5\}$ have been implemented according to their respective original formulations given in (1) and (7). For every experiment, we had an available RAM of 13.6 GB.

.2.1 Experiment 1

For this grid experiment, we considered three noise levels $\sigma^2 = \{0.01, 1, 100\}$, representing a small, medium and large noise regime, respectively. For each σ^2, we chose a maximum tolerance level of $\varepsilon_{\max} = 3.5$. For the small and medium noise regime, we chose a minimum tolerance level of $\varepsilon_{\min} = 0.05$, while for the large noise regime we chose $\varepsilon_{\min} = 2$. In fact, a tolerance level of 0.05 could not be achieved for any setting (α, β) when $\sigma^2 = 100$ and thus we chose a corresponding level of $\varepsilon_{\min} = 2$ for illustration purposes. For every noise level, we considered a linear grid $G_\beta = \{0, 0.05, 0.1, 0.15, 0.2..., 1\}$, where included the extreme case of $\beta = 1$. In particular, for $\sigma^2 = 0.01$, we considered a logarithmic grid $G_\alpha = \{-6, ..., 0.5\}, |G_\alpha| = 25$ and for $\sigma^2 = \{1, 100\}$, we considered a logarithmic grid $G_\alpha = \{-6, ..., 0.3\}, |G_\alpha| = 20$. Each grid experiment has been run with a maximum number of iterations $T_{\max} = 25000$ for three different seeds with SHB and SNAG started at $\boldsymbol{x}_1 = (0.2, ..., 0.2)^T \in \mathbb{R}^{20}$. We reported the median number of iterations T to reach an $\boldsymbol{x}_T$ with $f(\boldsymbol{x}_T) - f(\boldsymbol{x}^\star) \leq \varepsilon$ for each $\varepsilon \in \mathcal{E}$ over the three runs for every setting (α, β).

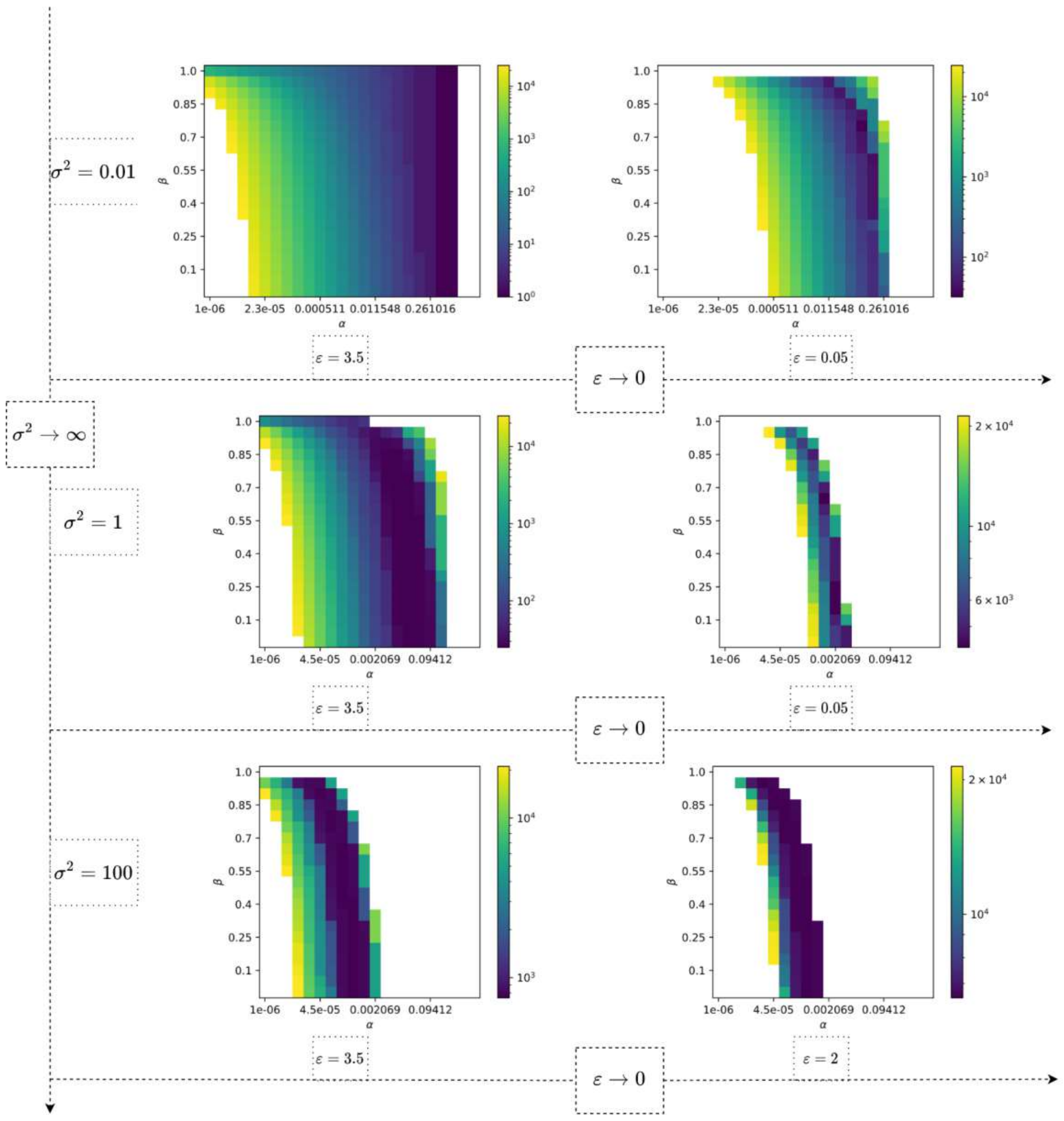


Figure 1: ***Experiment 1: SNAG*** We immediately observe that our conclusions from experiment 1, where we considered SHB, extend to the case of SNAG in the strongly convex setting.

.2.2 Experiment 2

For this experiment, we formulated a logistic regression model in PyTorch, employing one linear layer combined with the sigmoid activation function. We minimized the binary cross-entropy loss, `torch.nn.BCELoss`, training the model on the full diabetes dataset [7] with $n = 768$. We used two batch sizes $B = \{200, 10\}$, representing a small and large noise regime, respectively. For each B, we chose a maximum tolerance level of $\varepsilon_{\max} = 0.5$, while we picked different minimum tolerance levels, i.e., $\varepsilon_{\min} = 0.47$ for $B = 200$ and $\varepsilon_{\min} = 0.475$ for $B = 10$, for illustration purposes. In fact a tolerance level of 0.47 could not be achieved for any setting (α, β) when $B = 10$. For every noise level, we considered a linear grid $G_\beta = \{0, 0.05, 0.1, 0.15, 0.2..., 1\}$, where we included the extreme case of $\beta = 1$. For $B = 200$, we considered a logarithmic grid $G_\alpha = \{-4, ..., 0.4\}, |G_\alpha| = 20$ and for $B = 10$, we considered the logarithmic grid $G_\alpha = \{-3, ..., 0.8\}, |G_\alpha| = 20$. Each grid experiment has been run with a maximum number of epochs $E_{\max} = 1000$ for three different initialization points and three different seeds. We reported the median number of iterations to achieve a binary cross-entropy loss smaller or equal to every $\varepsilon \in \mathcal{E}$ for the three runs for every setting (α, β).

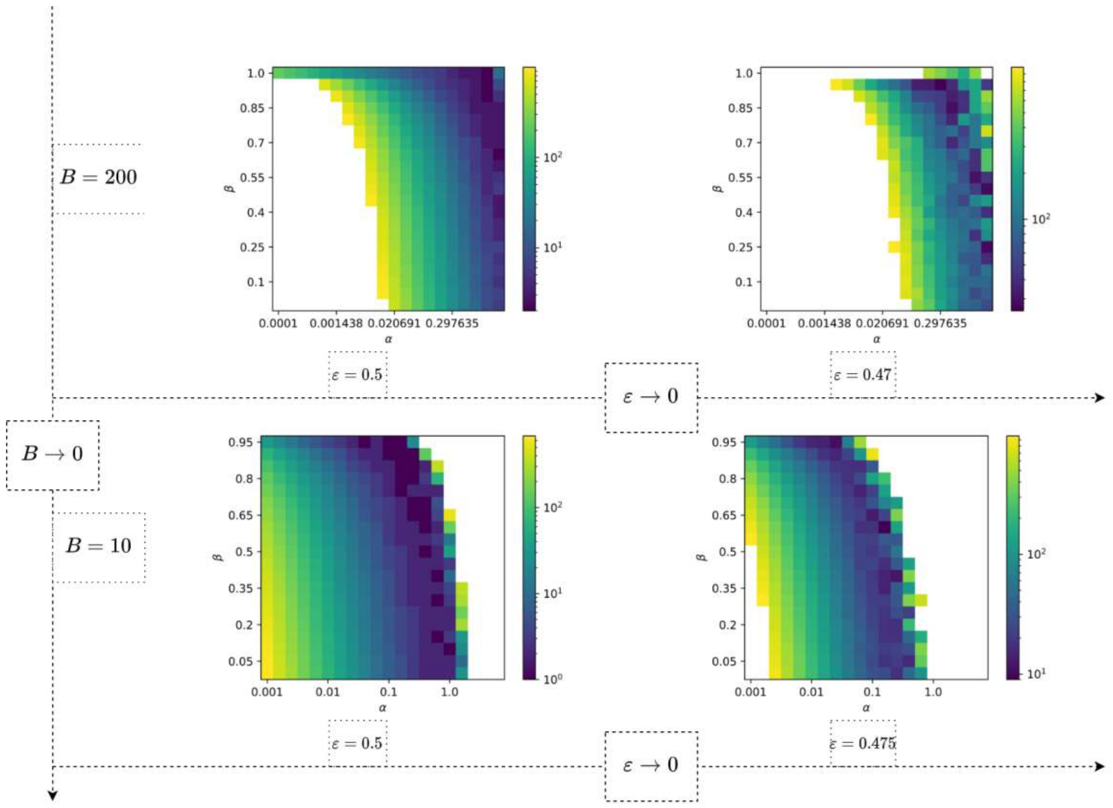
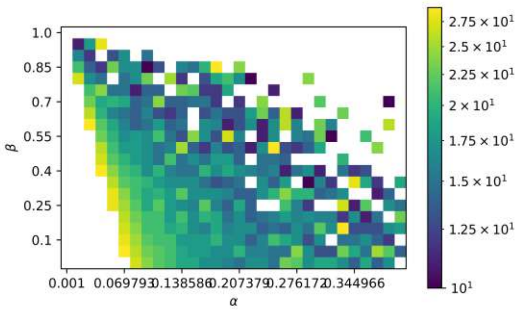


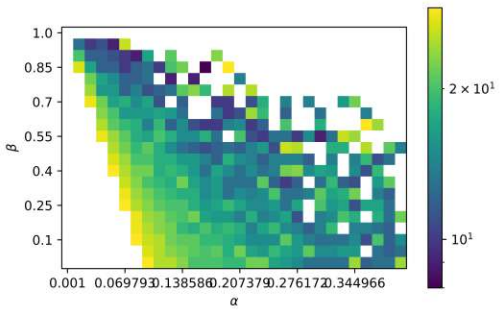
Figure 2: ***Experiment 2: SNAG*** We immediately observe that our conclusions from experiment 2, where we considered SHB, extend to the case of SNAG in the convex setting.

.2.3 Experiment 3

We used a PyTorch implementation of the neural network LeNet [24] which we trained on the Fashion-MNIST training dataset [56] ($n = 6 \cdot 10^4$) with a fixed batch size of $B = 256$. We considered different initialization points with different seeds for every grid experiment. With an epoch budget of $E_{\max} = 30$, we reported the number of epochs E to reach a cross-entropy loss, `torch.nn.CrossEntropyLoss`, of less than or equal to $\varepsilon = 0.23$ for every setting (α, β). We considered for both initializations a linear grid $G_\beta = \{0, 0.05, 0.1, 0.15, 0.2..., 1\}$, where we included the extreme case of $\beta = 1$, and a linear grid $G_\alpha = \{0.001, ..., 0.4\}, |G_\alpha| = 30$.



(a) initialization 1.



(b) initialization 2.

Figure 3: ***Experiment 3: SNAG*** As in experiment 3, where we considered SHB, we observe that the efficiency to converge for a setting (α, β) depends on the initialization point $\boldsymbol{x}_1$ for SNAG.

.2.4 Experiment 4

We considered two batch sizes $B = \{2560, 256\}$ with respective epoch budgets of $E_{\max} = \{10, 5\}$. As in experiment 3, we considered the neural network LeNet [24] which we trained on the Fashion-MNIST training dataset [56] with $n = 6 \cdot 10^4$. We fixed a momentum coefficient $\beta = 0.95$ and reported the final cross-entropy loss of SHB with (α, β) and plain SGD with $\tilde{\alpha}$ after $E_{\max}$ epochs using 100 different initialization points for $\alpha = \{0.005, 0.01, 0.02, 0.03\}$ which led to the effective learning rates for plain SGD of $\tilde{\alpha} = \{0.1, 0.2, 0.4, 0.6\}$. That is, for every initialization point $\boldsymbol{x}_{1j}, j = 1, ..., 100$, we ran SHB with every (α, β) and plain SGD with the corresponding $\tilde{\alpha}$ and reported the final cross-entropy loss after $E_{\max}$ epochs.

.2.5 Experiment 5

We considered three noise levels $\sigma^2 = \{0.01, 1, 100\}$, representing a small, medium and large noise regime, respectively, as well as three sharpness levels for the minimum $\{1, 10, 100\}$.The 1-dimensional Styblinski-Tang function has one global minimum at $x^\star = -2.903534$ and a saddle point at $s = 0.156731$ which separates the global minimum from a local minimum. We initialized SHB and SNAG at $\frac{1}{\sqrt{k}}(x^\star, ..., x^\star) \in \mathbb{R}^{20}$ and reported the number of iterations T to observe $\exists i \in [20] : x_{Ti} \geq s$, i.e., to observe one coordinate of an iterate $\boldsymbol{x}_T$ to escape the valley of the global minimum. Figure 4 illustrates the setup of this experiment for the 1-dimensional case.

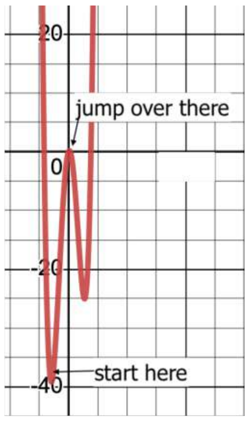


Figure 4: An illustration of experiment 5 for $p = 1$. We initialize SHB and SNAG at the global minimum and count the number of iterations to escape its valley.

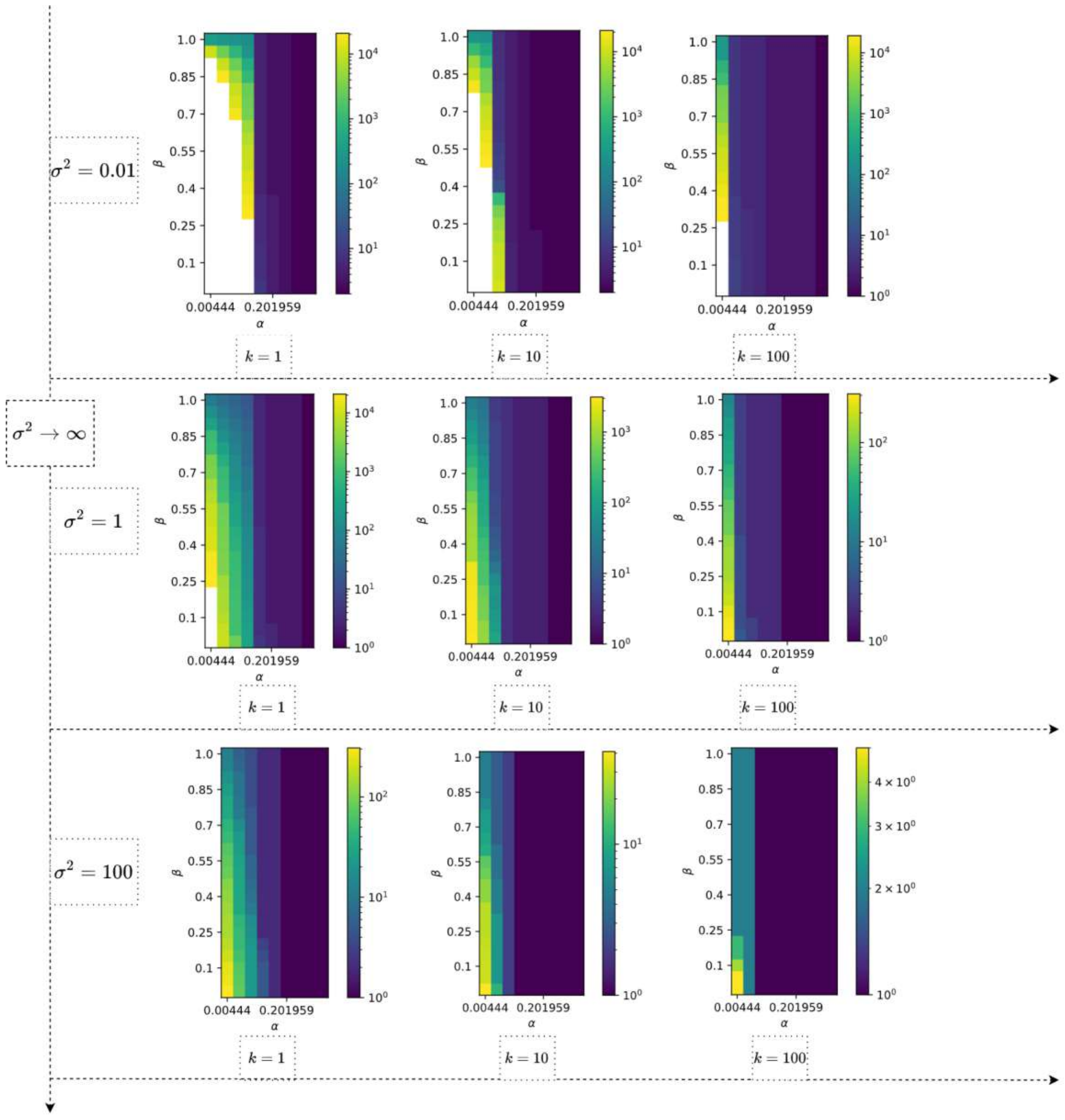


Figure 5: ***Experiment 5: SNAG*** We see that our conclusions from experiment 5, where we considered SHB, extend to the case of SNAG.

.2.6 Experiment 6

We subsampled the training dataset of Fashion-MNIST [56] by a factor of ≈ 0.0083, considering 500 images out of the $6 \cdot 10^4$ training images and found the local minimum $\bar{\boldsymbol{x}}_{GD}$ with GD, i.e., $B = 500$. We then employed a batch size of $B = 200$ which induces a large-batch setting with $B/n = 0.4$, and initialized SGDM at $\bar{\boldsymbol{x}}_{GD}$, where we were confronted with a sharpness estimate according to (3.1) with $M = 50, \delta = 0.01$ of ≈ 0.5. We reported the number of epochs E to reduce this initial estimate to a target value of $\varepsilon = 0.0025$ with an epoch budget of $E_{\max} = 50$ for every setting (α, β). We considered a linear grid $G_\beta = \{0.1, ..., 1\}, |G_\beta| = 10$, where we accounted for the extreme case of $\beta = 1$, and a linear grid $G_\alpha = \{0, , ..., 1\}, |G_\alpha| = 10$.

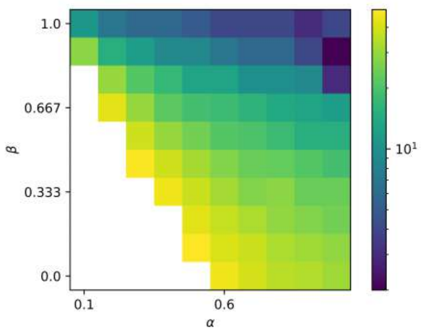


Figure 6: ***Experiment 6: SNAG*** We see that our conclusions from experiment 6, where we considered SHB, extend to the case of SNAG.

.2.7 Experiment 7

We trained the neural network ResNet-18 [12] on the CIFAR-10 [21] training set with $n = 5 \cdot 10^4$. For the small-batch benchmark, we trained the network with a setting of $(\alpha, \beta, B) = (0.1, 0, 100)$ for 200 epochs, where we employed a weight decay of 10^{-4} and decayed the learning rate α by 10 at epoch 150. With the goal of multiplying the batch size by a factor of $c = 20$, while keeping the remaining hyperparameters (epochs, weight decay, learning rate schedule, etc.) unchanged and maintaining a sensible level of SGN, the use of the LSR led us to multiply the learning rate α by 20 as well, inducing a new setting of $(2, 0, 2000)$. According to the MLSR, however, we employed a momentum coefficient of $\beta = 1 - \frac{1}{20} = 0.95$, leading to a new setting of $(0.1, 0.95, 2000)$. We reported the top-1 training and test error for every epoch and showed the error range of plus/minus two standard deviations for two runs.

.2.8 Experiment 8

We trained the neural network VGG11 [48] on the CIFAR-10 [21] training set with $n = 5 \cdot 10^4$. For the small-batch benchmark, we trained the network with a setting of $(\alpha, \beta, B) = (0.05, 0, 100)$ for 300 epochs, where we decayed the learning rate α by 2 at epochs $\{30, 60, 90, 120, 150, 180, 210, 240, 270\}$ and employed a weight decay of $5 \cdot 10^{-4}$. With the goal of multiplying the batch size by a factor of $c = 20$, while keeping the remaining hyperparameters (epochs, weight decay, learning rate schedule, etc.) unchanged and maintaining a sensible level of SGN, the use of the LSR led us to multiply the learning rate by 20 as well, inducing a new setting of $(1, 0, 2000)$. According to the MLSR, however, we employed a momentum coefficient of $\beta = 1 - \frac{1}{20} = 0.95$, leading to a new setting of $(0.05, 0.95, 2000)$. We reported the top-1 training and test error for every epoch and showed the error range of plus/minus two standard deviations for two runs.

.2.9 Experiment 9

We trained the neural network ResNet-18 [12] on the CIFAR-10 [21] training set ($n = 5 \cdot 10^4$), where we employed a weight decay of 10^{-4}. We considered a small-batch benchmark of training the network with $(\alpha, \beta, B) = (0.05, 0, 128)$, reporting the top-1 test error after 10 epochs. We then applied both the LSR and the MLSR to the given the set of batch size scalings $\{2, 4, 8, 16\}$. We again reported the top-1 test error after 10 epochs, keeping all remaining hyperparameters (weight decay, learning rate schedule, etc.) unchanged. This procedure yielded the following settings (α, β, B) for the batch size scalings:

- $c = 2 \rightsquigarrow$ LSR:$(0.1, 0, 256)$, MLSR:$(0.05, 0.5, 256)$,
- $c = 4 \rightsquigarrow$ LSR:$(0.2, 0, 512)$, MLSR:$(0.05, 0.75, 512)$,
- $c = 8 \rightsquigarrow$ LSR:$(0.4, 0, 1024)$, MLSR:$(0.05, 0.875, 1025)$,
- $c = 16 \rightsquigarrow$ LSR:$(0.8, 0, 2048)$, MLSR:$(0.05, 0.9375, 2048)$.

We showed the error range of plus/minus two standard deviations for three runs.