Amortized Nesterov's Momentum: A Robust Momentum and Its Application to Deep Learning

Kaiwen Zhou[1] **Yanghua Jin**[2] **Qinghua Ding**[1] **James Cheng**[1]
[1] Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2] Preferred Networks, Inc.
kwzhou@cse.cuhk.edu.hk, jinyh@preferred.jp,
qhding@cse.cuhk.edu.hk, jcheng@cse.cuhk.edu.hk

Abstract

This work proposes a novel momentum technique, the *Amortized Nesterov's Momentum*, for stochastic convex optimization. The proposed method can be regarded as a smooth transition between Nesterov's method and mirror descent. By tuning only a single parameter, users can trade Nesterov's acceleration for robustness, that is, the variance control of the stochastic noise. Motivated by the recent success of using momentum in deep learning, we conducted extensive experiments to evaluate this new momentum in deep learning tasks. The results suggest that it can serve as a favorable alternative for Nesterov's momentum.

1 INTRODUCTION

In convex optimization, momentum methods have been widely adopted to minimize an objective function $f : \mathbb{R}^n \to \mathbb{R}$, and recently they have been extended into training deep neural networks. The most notable momentum techniques are classical momentum (Polyak, 1964) and Nesterov's momentum (Nesterov, 1983, 2013b). Between these two, Nesterov's method is famous for its optimal convergence rates for a wider range of convex problems, and it has the following scheme[1] (constant step) (Nesterov, 2013b) ($y \in \mathbb{R}^n, \eta, \beta \in \mathbb{R}$ and $y_0 = x_0$):

$$\begin{aligned} y_{k+1} &= x_k - \eta \cdot \nabla f(x_k), \\ x_{k+1} &= y_{k+1} + \beta \cdot (y_{k+1} - y_k), \text{ for } k \geq 0, \end{aligned} \tag{1}$$

where $\nabla f(x_k)$ is the gradient of f at x_k and we call $\beta \cdot (y_{k+1} - y_k)$ the momentum. Note that if $\beta = 0$, scheme (1) reduces to gradient descent (GD). Thus, we can regard scheme (1) as injecting the momentum into the sequence of GD, and the momentum comes from the previous iterate generated by GD.

The setting where the gradient oracle can be inexact arises naturally in many optimization tasks, and it has been extensively studied for decades. If the inexactness is stochastic, stochastic gradient descent (SGD) (Robbins and Monro, 1951) is a classical choice of optimizer, and due to its low iteration cost, SGD has also been widely used for deep learning tasks. Under the classic bounded noise setting, notable schemes have been proposed: Robust SA (and its generalized variant, mirror descent SA) (Nemirovski et al., 2009), which is a more robust SA approach; AC-SA (Lan, 2012), which incorporates Nesterov-style acceleration into SA and achieves the optimal rate in the convex setting. In the constant step case, AC-SA is equivalent to the scheme (1) with $\nabla f(x_k)$ replaced by a stochastic gradient. Such a scheme is called the SGD with Nesterov's momentum, which is very popular in training deep neural networks.

There are also recent studies which focus on momentum acceleration in different stochastic settings that allow unbounded noise: Yuan et al. (2016) questioned the benefit of momentum in the constant step case and proposed to use decaying momentum. Vaswani et al. (2019) leveraged interpolation-like conditions (Ma et al., 2018; Belkin et al., 2018) and proposed a parameter setting for stochastic Nesterov's method which recovers the accelerated rates. Jain et al. (2018); Kidambi et al. (2018); Liu and Belkin (2020) studied the non-acceleration issues of stochastic momentum schemes on least squares and proposed new schemes that converge faster than SGD.

If the gradient noise is deterministic, Devolder et al. (2014) noted that under their notion of noise, Nesterov's method can lead to an accumulation of error and diverge while GD is much more robust to the noise. Lessard et al. (2016) proposed to use robust control theory to study the trade-off between robustness and convergence rate when the gradient is subject to multiplicative noise. Inspired

[1] We exchange the notations of x and y in Nesterov (2013b).

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

by their work, Cyrus et al. (2018) proposed the Robust Momentum Method (RMM), which has a parameter that can be tuned to trade convergence rate for robustness. RMM can be regarded as a smooth transition between an accelerated scheme and GD.

In this work, we propose a novel momentum technique, the *Amortized Nesterov's Momentum*. Similar to RMM, it introduces a parameter to trade Nesterov's acceleration (not convergence rate) for robustness, while our notion of robustness[2] is for stochastic gradient noise. At one extreme, the proposed method is equivalent to AC-SA and enjoys the optimal rate. At the other extreme, the method becomes mirror descent SA, which has a constant-factor better variance control than AC-SA. Unlike RMM, our trade-off does not necessarily lead to a slower convergence rate and our technique has clearer intuition.

High-level idea: stochastic Nesterov's momentum can be unreliable since it is provided only by the previous iterate. The iterate potentially has large variance, which may lead to a false momentum that perturbs the training process. We thus propose to use the stochastic Nesterov's momentum based on several past iterates, which provides robust acceleration. In other words, instead of immediately using an iterate to provide momentum, we put the iterate into an "amortization plan" and use it later.

We analyze the proposed methods in a general setting that covers smooth/non-smooth, deterministic/stochastic convex problems and allows choosing non-Euclidean norm for the problem space.

We mainly focus on evaluating this new momentum in deep learning, which is motivated by the following facts:

- Sutskever et al. (2013) found that using SGD with Nesterov's momentum achieves substantial speedups for training neural networks, which essentially turns it into the benchmarking method of neural network design, especially for classification tasks (He et al., 2016a,b; Zagoruyko and Komodakis, 2016; Huang et al., 2017).
- Ma and Yarats (2019) proposed the QHM method, which shows good performance for deep learning tasks. QHM can be regarded as a reformulation of RMM (see Appendix C.4 in Ma and Yarats (2019)).

We conducted extensive deep learning experiments to explore the benefits of this new momentum, which shows that it can serve as a nice alternative for Nesterov's momentum. We also conducted convex experiments (in Appendix A.8) as sanity checkers for the theoretical results.

[2] In this work, robustness refers to how well the method controls the variance of the stochastic noise, i.e., the variance term in the expected error and the probability of large deviations.

2 PRELIMINARIES

Notations and generalities We use E to denote a finite-dimensional real vector space and E^* is its dual space. The value of a linear function $g \in E^*$ at $x \in E$ is represented by $\langle g, x\rangle$. $\|\cdot\|$ denotes an arbitrary norm in E and the dual norm $\|\cdot\|_*$ on E^* is defined in the standard way: $\|g\|_* \triangleq \max_{\|x\|=1} \langle g, x\rangle$. Scalar multiplication for $v \in E$ and $\beta \in \mathbb{R}$ is denoted as $\beta \cdot v$. The notation $[m]$ refers to the set $\{1, \dots, m\}$ and the symbol $\leftarrow$ denotes assignment. We use $\mathbb{E}$ to denote expectation and the conditional expectation for a random process $i_0, i_1, \dots$ is denoted as $\mathbb{E}_{i_k}[\cdot] \triangleq \mathbb{E}[\cdot \mid (i_0, \dots, i_{k-1})]$.

Problem setup We consider the convex composite problem (Beck and Teboulle, 2009; Nesterov, 2013a): $\min_{x\in X}\{F(x) \triangleq f(x) + h(x)\}$, where $X \subseteq E$ is a non-empty closed convex set and h is a proper convex function. We denote $x^\star \in X$ as a solution to this problem. $\nabla f(x) \in E^*$ represents (one of) the (sub)gradient of f at x. Given an input $x \in E$, the stochastic gradient oracle outputs an unbiased $\nabla f_i(x) \in E^*$, where the random variable i is independent of x.

We introduce the proximal setting, which generalizes the usual Euclidean setting. The *distance generating function* $d : X \to \mathbb{R}$ is required to be continuously differentiable and 1-strongly convex with respect to $\|\cdot\|$, i.e., $d(x) - d(y) - \langle \nabla d(y), x - y\rangle \geq \frac{1}{2}\|x - y\|^2, \forall x, y \in X$. The *prox-term* (*Bregman divergence*) associated with d is $V_d(x, y) \triangleq d(x) - d(y) - \langle \nabla d(y), x - y\rangle, \forall x, y \in X$. By adjusting $\|\cdot\|$ and $d(\cdot)$ to the geometry of the problem, mirror descent achieves a smaller problem-dependent constant than the Euclidean algorithms, which is its key benefit (Nemirovski and Yudin, 1983). Typical proximal setups can be found in Section 5.3.3 in Ben-Tal and Nemirovski (2013). At a first reading, this setting can be taken as the standard Euclidean setting: $X = E = \mathbb{R}^n$, $\|\cdot\| = \|\cdot\|_2$, $\langle\cdot,\cdot\rangle$ is the inner product, $d(x) = \frac{1}{2}\|x\|_2^2$ and $V_d(x, y) = \frac{1}{2}\|x - y\|_2^2$.

We assume that V_d is chosen such that the *prox-mapping*, $\mathrm{Prox}_h(x, \mathcal{G}) \triangleq \arg\min_{u\in X}\{V_d(u, x) + \langle \mathcal{G}, u\rangle + h(u)\}$, can be easily computed for any $x \in X, \mathcal{G} \in E^*$. Examples where this assumption is satisfied can be found in Parikh et al. (2014); Ghadimi and Lan (2012).

3 AMORTIZED NESTEROV'S MOMENTUM

In this section, we introduce SGD with Amortized Nesterov's Momentum (AM1-SGD) in Algorithm 1, and in Algorithm 2, we reformulate Algorithm 1 into a "momentum scheme" under the Euclidean setting with $h \equiv 0$

Alg. 1 AM1-SGD

Input: Initial guess x_0, parameter $\{\alpha_s\}$, momentum $\{\beta_s\}$, amortization length m, iteration number K.

Initialize: $\tilde{x}_0 = z_0 = x_0, S = K/m$.[3]

for $s = 0, \ldots, S-1$ **do**
  for $j = 0, \ldots, m-1$ **do**
    $k = sm + j$.
    $x_k = (1-\beta_s) \cdot z_k + \beta_s \cdot \tilde{x}_s$.
    $z_{k+1} = \mathrm{Prox}_{\alpha_s h}\left(z_k, \alpha_s \cdot \nabla f_{i_k}(x_k)\right)$.
  end for
  $\tilde{x}_{s+1} = \frac{1-\beta_s}{m} \cdot \sum_{j=1}^{m} z_{sm+j} + \beta_s \cdot \tilde{x}_s$.
end for

Output: $\tilde{x}_S$.

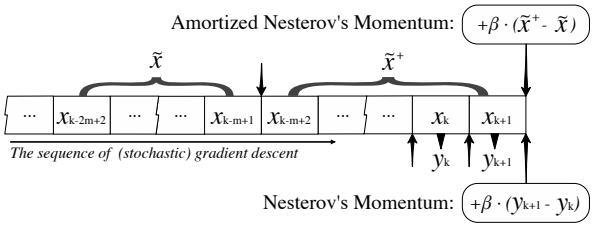


Figure 1: Graphical illustration of Amortized Nesterov's Momentum. This figure describes how the momentum is injected into the sequence of gradient descent $\{x_k\}$.

and constant momentum $\beta_s = \beta$. We provide the step-by-step conversion in Appendix B.1, which shows that Algorithms 1 and 2 are equivalent through $\eta = \alpha_s(1-\beta_s)$. This momentum scheme is related to how we implement AM1-SGD for deep learning applications and is also clearer for providing intuition. In Section 3.1, we propose another method (AM2-SGD) to implement the idea of utilizing several past iterates. To elaborate the features of AM1-SGD, we make the following remarks:

A periodical and large momentum A graphical illustration of Algorithm 2 is included in Figure 1, which depicts how AM1-SGD leverages several past iterates to provide momentum. Nesterov's momentum is injected in every iteration. In comparison, the amortized momentum is injected every m iterations, while this momentum $\beta \cdot (\tilde{x}^+ - \tilde{x})$ is expected to be much larger than $\beta \cdot (y_{k+1} - y_k)$ if the same η and β are used. Intuitively, we can understand the amortized momentum as an m times larger Nesterov's momentum, which is applied every m iterations.

The bridge between accelerated schemes and mirror descent It can be verified that if $m = 1$, Algorithm 2 is equivalent to (stochastic) Nesterov's scheme (1) and Algorithm 1 becomes AC-SA (Lan, 2012); if $m = K$, Algorithm 2 is the SGD that outputs the average of the whole history and Algorithm 1 is equivalent to mirror descent SA (Nemirovski et al., 2009; Lan, 2012).

[3]For simplicity, we assume K is divisible by m.

Alg. 2 AM1-SGD (Euclidean, $h \equiv 0$, constant scheme)

Input: Initial guess x_0, learning rate η, momentum β, amortization length m, iteration number K.

Initialize: $x \leftarrow x_0, \tilde{x} \leftarrow x_0, \tilde{x}^+ \leftarrow \mathbf{0}$.

for $k = 0, \ldots, K-1$ **do**
  $x \leftarrow x - \eta \cdot \nabla f_{i_k}(x)$.
  $\tilde{x}^+ \leftarrow \tilde{x}^+ + \frac{1}{m} \cdot x$.
  if $(k+1) \mod m = 0$ **then**
    // adding amortized momentum.
    $x \leftarrow x + \beta \cdot (\tilde{x}^+ - \tilde{x})$.
    $\tilde{x} \leftarrow \tilde{x}^+, \tilde{x}^+ \leftarrow \mathbf{0}$.
  end if
end for

Output: Option I: x, Option II: $\tilde{x}$.

Acceleration and tail averaging The main ingredients of AM1-SGD are Nesterov-style acceleration and tail averaging, namely, the output point $\tilde{x}$ is an m-iterations tail average and the amortized momentum is provided by two consecutive tail averages. It seems that the effects of outputting a tail average and applying the amortized momentum are independent. Option I in Algorithm 2, which we provide as a heuristic option, omits the tail averaging at the output point.

Options Option II in Algorithm 2, which corresponds to the output of Algorithm 1, is the theoretical option that we analyze in Section 4. Option I, in addition to omitting the tail averaging effect, follows the implementations of Nesterov's momentum in PyTorch (Paszke et al., 2017) and Tensorflow (Abadi et al., 2016). We will see in Section 5 that the standard Nesterov's momentum also has this type of heuristic and theoretical options.

Connections with Katyusha Our original inspiration of AM1-SGD comes from Katyusha (Allen-Zhu, 2018), the recent breakthrough in finite-sum convex optimization, which uses a previously calculated "snapshot" point to provide momentum, i.e., Katyusha momentum. AM1-SGD also uses an aggregated point to provide momentum and it shares many structural similarities with Katyusha. We refer interested readers to Appendix B.3.

3.1 AM2-SGD

We propose another realization of the amortization technique (AM2-SGD) in Algorithm 3, and similar to AM1-SGD, its "momentum scheme" reformulation in Algo-

Alg. 3 AM2-SGD

Input: Initial guess x_0, amortization length m, a point table $\phi = \begin{bmatrix}\phi_1 & \cdots & \phi_m\end{bmatrix} \in E^m$, parameter $\{\alpha_k\}$, momentum $\{\beta_k\}$, iteration number K.

Initialize: $z_0 = \phi_j^0 = x_0, \forall j \in [m]$.

1: **for** $k = 0, \dots, K-1$ **do**
2: Sample j_k uniformly in $[m]$.
3: $x_k^{j_k} = (1-\beta_k)\cdot z_k + \beta_k \cdot \phi_{j_k}^k$.
4: $z_{k+1} = \mathrm{Prox}_{\alpha_k h}\left(z_k, \alpha_k \cdot \nabla f_{i_k}(x_k^{j_k})\right)$.
5: $\phi_{j_k}^{k+1} = (1-\beta_k)\cdot z_{k+1} + \beta_k \cdot \phi_{j_k}^k$ and keep other entries unchanged (i.e., $\phi_j^{k+1} = \phi_j^k$ for $j \neq j_k$).
6: **end for**

Output: $\bar{\phi}^K = \frac{1}{m}\sum_{j=1}^m \phi_j^K$.

Alg. 4 AM2-SGD (Euclidean, $h \equiv 0$, constant scheme)

Input: Initial guess x_0, amortization length m, a point table $\phi \in \mathbb{R}^{n\times m}$, learning rate η, momentum β, iteration number K.

Initialize: $\phi_j^0 = x_0, \forall j \in [m]$. j_0 is uniformly sampled in $[m]$. If Option II, store a running average $\bar{\phi}^0 = x_0$.

1: **for** $k = 0, \dots, K-1$ **do**
2: $\phi_{j_k}^{k+1} = x_k - \eta \cdot \nabla f_{i_k}(x_k)$ and keep other entries unchanged (i.e., $\phi_j^{k+1} = \phi_j^k$ for $j \neq j_k$).
3: Sample j_{k+1} uniformly in $[m]$.
4: $x_{k+1} = \phi_{j_k}^{k+1} + \beta \cdot (\phi_{j_{k+1}}^{k+1} - \phi_{j_k}^k)$.
5: **if** Option II **then** $\bar{\phi}^{k+1} = \bar{\phi}^k + \frac{1}{m}\cdot\left(\phi_{j_k}^{k+1} - \phi_{j_k}^k\right)$.
6: **end for**

Output: Option I (unstable): x_K, Option II: $\bar{\phi}^K$.

rithm 4. We were inspired by the constructions of SVRG (Johnson and Zhang, 2013) and SAGA (Defazio et al., 2014), the most popular methods in finite-sum convex optimization—to reuse the information from several past iterates, we can either maintain a "snapshot" that aggregates the information or keep the iterates in a table.

We discuss some interesting characteristics of AM2-SGD by making the following remarks:

Identical iterations The workload of AM1-SGD varies for different iterations due to the if-clause (or the two-loop structure). This to some extent limits its extensibility to other settings (e.g., asynchronous setting). AM2-SGD does not have this issue and is structurally simpler. Although AM2-SGD requires storing a table of vectors, which could be expensive in practice, the table size m is tunable, and we will see that in theory, it is more beneficial to choose relatively small m.

"Random tail averaging" Based on the expectation of geometric distribution, we know that the point table ϕ is expected to store m iterates from the most recent $\Theta(m \log m)$ iterates. Thus, we can regard the output $\bar{\phi}$, the average of the point table, as a "random tail average". The momentum of AM2-SGD is randomly provided by two past iterates in the table. Interestingly, as shown in Section 6.2, when using the same (η, β, m), the convergence of AM2-SGD is similar to AM1-SGD while being slightly faster. This suggests that randomly incorporating past iterates beyond m iterations helps.

Options As is the case for AM1-SGD, we provide Option I in Algorithm 4 following the same heuristics. However, in our preliminary experiments, we found that the performance of Option I is not stable, and thus we do not recommend this option for AM2-SGD. We believe that it is caused by the additional randomness $\{j_k\}$.

4 CONVERGENCE RESULTS

In this section, we analyze AM1-SGD (Algorithm 1) and AM2-SGD (Algorithm 3) in the convex setting. Comparing Algorithms 1 and 3, we see that their iterations can be generalized as follows ($y^+ = x_{k+1}$ for AM1-SGD):

$$\begin{aligned} x &= (1-\beta)\cdot z + \beta \cdot y, \\ z^+ &= \mathrm{Prox}_{\alpha h}\left(z, \alpha \cdot \nabla f_i(x)\right), \\ y^+ &= (1-\beta)\cdot z^+ + \beta \cdot y. \end{aligned} \tag{2}$$

This scheme is first proposed in Auslender and Teboulle (2006), which represents one of the simplest variants of Nesterov's methods (see Tseng (2008) for the others). This scheme is modified into various settings (Hu et al., 2009; Lan, 2012; Ghadimi and Lan, 2012; Zhou et al., 2018, 2019; Lan et al., 2019) to achieve acceleration.

We impose the following assumptions on the regularity of f and ∇f_i, which are classical in the analysis of stochastic approximation algorithms (identical to the ones in Ghadimi and Lan (2012) with $\mu = 0$):

Assumptions. *For some $L \geq 0, M \geq 0, \sigma \geq 0$,*

(a) $0 \leq f(y) - f(x) - \langle \nabla f(x), y - x\rangle \leq \frac{L}{2}\|y-x\|^2 + M\|y-x\|, \forall x, y \in X$.[4]

(b) $\mathbb{E}_i\left[\nabla f_i(x)\right] = \nabla f(x), \forall x \in X$.

(c) $\mathbb{E}_i\left[\|\nabla f_i(x) - \nabla f(x)\|_*^2\right] \leq \sigma^2, \forall x \in X$.

These assumptions cover several important classes of convex problems. For example, *(a)* covers the cases of f being L-smooth ($M = 0$) or L_0-Lipschitz continuous ($M = 2L_0, L = 0$) convex functions and if $\sigma = 0$ in

[4] When $M > 0$, f is not necessarily differentiable and $\nabla f(x)$ denotes an arbitrary subgradient of f at x.

(c), the assumptions cover several classes of deterministic convex programming problems.

The following lemma serves as a cornerstone for the convergence analysis of AM1-SGD and AM2-SGD. All the proofs in this paper are given in Appendix B.2.

Lemma 1. *Let* $\delta_x \triangleq \nabla f(x) - \nabla f_i(x)$. *If* $\alpha(1-\beta) < \frac{1}{L}$, *the update scheme (2) satisfies the recursion:*

$$\begin{aligned}&\frac{1}{1-\beta}\big(F(y^+) - F(x^\star)\big) + \frac{1}{\alpha}V_d(x^\star, z^+)\\ \le\ &\frac{\beta}{1-\beta}\big(F(y) - F(x^\star)\big) + \frac{1}{\alpha}V_d(x^\star, z)\\ &+ \frac{(\|\delta_x\|_* + M)^2}{2(\alpha^{-1} - L(1-\beta))} + \langle \delta_x, z - x^\star\rangle\,.\end{aligned}$$

Based on this key recursion, we establish the convergence rates for AM1-SGD and AM2-SGD as follows.

Theorem 1. *In AM1-SGD, if* $\beta_s = \frac{s}{s+2}$, $\alpha_s = \frac{\lambda_1}{L(1-\beta_s)}$ *where* $\lambda_1 = \min\left\{\frac{2}{3}, \frac{L\sqrt{V_d(x^\star,x_0)}}{\sqrt{2m}\sqrt{\sigma^2+M^2}(S+1)^{\frac{3}{2}}}\right\}$. *Then,*

(a) The output $\tilde{x}_S$ *satisfies*

$$\begin{aligned}&\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star) \le \frac{6LmV_d(x^\star, x_0)}{(K+m)^2}\\ &+ \frac{8\sqrt{2V_d(x^\star,x_0)}\sqrt{\sigma^2+M^2}}{\sqrt{K+m}} \triangleq \mathcal{K}_0(m).\end{aligned}$$

(b) If X *is compact and the variance has a "light tail", i.e.,* $\mathbb{E}_i\left[\exp\left\{\|\nabla f_i(x) - \nabla f(x)\|_*^2/\sigma^2\right\}\right] \le \exp\{1\}$, $\forall x \in X$, *denoting* $D_X \triangleq \max_{x\in X}\|x - x^\star\|$, *for any* $\Lambda \ge 0$, *we have*

$$\begin{aligned}&\mathrm{Prob}\left\{F(\tilde{x}_S) - F(x^\star) \le \mathcal{K}_0(m) + \mathcal{K}_1(m,\Lambda)\right\}\\ &\ge 1 - \left(\exp\{-\Lambda^2/3\} + \exp\{-\Lambda\}\right),\end{aligned}$$

where the deviation term $\mathcal{K}_1(m,\Lambda)$ *is*

$$\mathcal{K}_1(m,\Lambda) \triangleq \frac{4\sqrt{6}\Lambda\sigma\left(\sqrt{3V_d(x^\star,x_0)} + D_X\right)}{3\sqrt{K+m}}.$$

Remark (a): Theorem 1a gives the expected objective error, from which the trade-off of m is clear: Increasing m improves the dependence on variance σ but deteriorates the $O(L/K^2)$ term (i.e., the acceleration). Note that for AM1-SGD, m is strictly constrained in $\{1,\ldots,K\}$. When $m = K$, AM1-SGD is equivalent to mirror descent SA, and the convergence rate in Theorem 1a becomes the corresponding $O(L/K + (\sigma+M)/\sqrt{K})$ (cf. Theorem 1 in Lan (2012)). By taking derivative, we see that the minimum of the expected error $\mathcal{K}_0(m)$ is obtained at either $m = 1$ or $m = K$. This to some extent undermines the choices of setting $1 < m < K$. However, it is worth noting that in practice, the values $V_d(x^\star, x_0)$, σ, L and M could be unknown, especially $V_d(x^\star, x_0)$. In this case, these values are chosen as some upper estimations and can be very inaccurate. The parameter m allows users to determine the amount of acceleration and variance control for concrete tasks, which is much more flexible than sticking to $m = 1$ or $m = K$.

Remark (b): Theorem 1b provides the probability of the objective value deviating from its expected performance (i.e., $\mathcal{K}_0(m)$). It is clear that increasing m leads to smaller deviations with the same probability and thus improves the robustness of the iterates. The additional compactness and "light tail" assumptions are similarly required in Nemirovski et al. (2009); Lan (2012); Ghadimi and Lan (2012). Note that the "light tail" assumption is stronger than Assumption (c). Recently, Nazin et al. (2019) established similar bounds without the "light tail" assumption by truncating the gradient. However, as indicated by the authors, their technique cannot be used for accelerated algorithms due to the accumulation of bias.

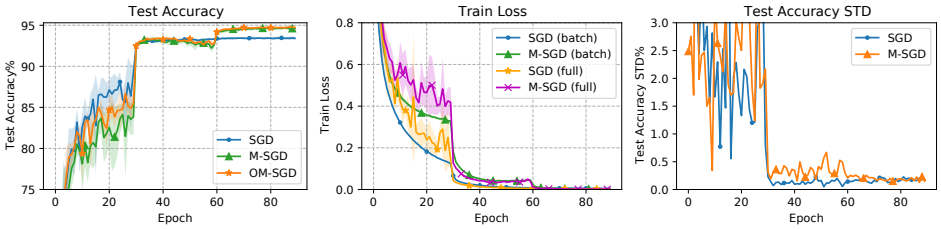
For AM2-SGD, we only give the expected convergence results as follows.

Theorem 2. *In AM2-SGD, if* $\beta_k = \frac{k/m}{k/m+2}$ *and* $\alpha_k = \frac{\lambda_2}{L(1-\beta_k)}$ *where* $\lambda_2 = \min\left\{\frac{2}{3}, \frac{L\sqrt{V_d(x^\star,x_0)}}{\sqrt{m}(\sigma+M)\left(\frac{K-1}{m}+2\right)^{\frac{3}{2}}}\right\}$, *the output* $\bar{\phi}^K$ *satisfies*

$$\begin{aligned}&\mathbb{E}\left[F(\bar{\phi}^K)\right] - F(x^\star)\\ \le\ &\frac{4(m^2-m)\big(F(x_0) - F(x^\star)\big) + 6LmV_d(x^\star,x_0)}{(K+2m-1)^2}\\ &+ \frac{8\sqrt{V_d(x^\star,x_0)}(\sigma+M)}{\sqrt{K+2m-1}}.\end{aligned} \tag{3}$$

Remark: In comparison with Theorem 1a, Theorem 2 has an additional term $F(x_0) - F(x^\star)$ in the upper bound, which is inevitable. This difference comes from different restrictions on the choice of m. For AM2-SGD, $m \ge 1$ is the only requirement. Since it is impossible to let $m \gg K$ to obtain an improved rate, this additional term is inevitable. As a sanity check, we can let $m \to \infty$ to obtain a point table with almost all x_0, and then the upper bound becomes exactly $F(x_0) - F(x^\star)$. Since the first term in (3) increases rapidly with m, a smaller m is favored for AM2-SGD. In some cases, there exists an optimal choice of $m > 1$ in Theorem 2. However, the optimal choice could be messy and thus we omit the discussion here. Comparing the rates, we see that when using the same m, AM2-SGD has slightly better dependence on σ, which is related to the observation in Section 6.2 that AM2-SGD is always slightly faster than AM1-SGD.

If $m = O(1)$, Theorems 1 and 2 establish the optimal



(a) (b) (c)

METHOD	Avg. STD
SGD	0.99%
M-SGD	1.04%

Figure 2: ResNet34 on CIFAR-10. For all methods, initial learning rate $\eta_0 = 0.1$, momentum $\beta = 0.9$, run 5 seeds (start at same x_0). In (a) (b), we plot mean curves with shaded bands indicate ± 1 standard deviation. (c) shows the standard deviation of test accuracy and its average over 90 epochs.

$O(L/K^2+(\sigma+M)/\sqrt{K})$ rate in the convex setting (see Lan (2012) for optimality), which verifies AM1-SGD and AM2-SGD as variants of Nesterov's method (Nesterov, 1983, 2013b). Note that the improvements on step-size policy proposed in Hu et al. (2009) and Ghadimi and Lan (2012) are orthogonal to the amortization technique and thus can be directly used in Theorems 1 and 2. From the above analysis, the effect of m can be understood as trading Nesterov's acceleration (the $O(1/K^2)$ term) for variance control (the $O(1/K)$ term). Since the convergence speed is determined by the sum of these two terms, the practical effect of amortization on convergence is undetermined. We expect amortization boosts the rate if σ or M is large, which is justified in Appendix A.8.

5 USING AMORTIZED NESTEROV'S MOMENTUM IN DEEP LEARNING

We start with reviewing the usage of Nesterov's momentum in deep learning. We discuss some subtleties in the implementation and evaluation, which contributes to the interpretation of our methods.

By replacing $\nabla f(x_k)$ with $\nabla f_{i_k}(x_k)$ in scheme (1), we obtain the SGD with Nesterov's momentum,

$$\begin{aligned} y_{k+1} &= x_k - \eta \cdot \nabla f_{i_k}(x_k), \\ x_{k+1} &= y_{k+1} + \beta \cdot (y_{k+1} - y_k), \text{ for } k \geq 0, \end{aligned} \tag{4}$$

which is widely used in deep learning. To make this point clear, recall that the reformulation in Sutskever et al. (2013) (scheme (5), also the Tensorflow version) and the PyTorch version (scheme (6)) have the following schemes ($v, v^{pt} \in \mathbb{R}^n$ and $v_0 = v_0^{pt} = \mathbf{0}$): for $k \geq 0$,

$$\begin{aligned} \textit{TF:} \quad v_{k+1} &= \beta \cdot v_k - \eta \cdot \nabla f_{i_k}(y_k + \beta \cdot v_k), \\ y_{k+1} &= y_k + v_{k+1}. \end{aligned} \tag{5}$$

$$\begin{aligned} \textit{PT:} \quad v_{k+1}^{pt} &= \beta \cdot v_k^{pt} + \nabla f_{i_k}(x_k), \\ x_{k+1} &= x_k - \eta \cdot (\beta \cdot v_{k+1}^{pt} + \nabla f_{i_k}(x_k)). \end{aligned} \tag{6}$$

Here the notations are modified based on their equivalence to scheme (4). It can be verified that schemes (5) and (6) are equivalent to (4) through $v_k = \beta^{-1} \cdot (x_k - y_k)$ and $v_k^{pt} = \eta^{-1}\beta^{-1} \cdot (y_k - x_k)$, respectively (see Defazio (2019) for other equivalent forms of scheme (4)).

Interestingly, both PyTorch and Tensorflow[5] track $\{x_k\}$, which we refer to as **M-SGD**. This choice allows a consistent implementation when wrapped in a generic optimization layer (Defazio, 2019). The accelerated rate is built upon $\{y_k\}$ in Nesterov (2013b). We use **OM-SGD** to refer to the Original M-SGD that outputs $\{y_k\}$.

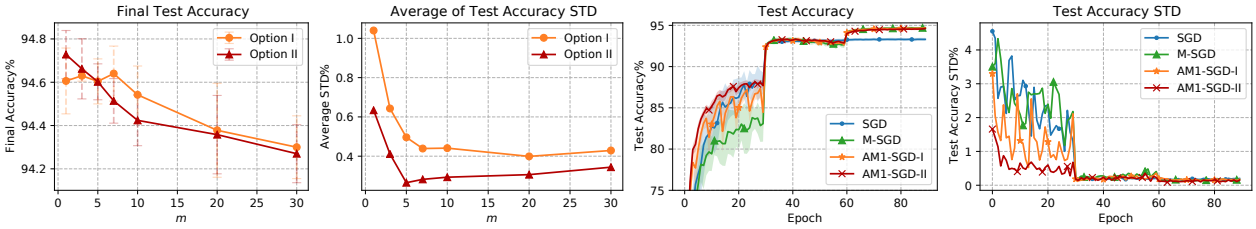
It can be verified that if $m = 1$, AM1-SGD (Algorithm 2) and AM2-SGD (Algorithm 4) with Option I are equivalent to M-SGD, and with Option II, they are equivalent to OM-SGD. By slightly modifying Algorithm 2, we can reduce its amortized iteration cost. We discuss this and other implementation details in Appendix A.1.

To introduce some evaluation metrics, we report the results of training ResNet34[6] (He et al., 2016a) on CIFAR-10 (Krizhevsky et al., 2009) using SGD and M-SGD in Figure 2 and make the following remarks:

- *The role of SGD.* The performance of SGD is used as a reference in this paper. Relating to Figure 1, we regard momentum as an add-on to plain SGD, and thus we choose the same learning rates for SGD and the momentum schemes. Such a perspective helps us understand what has been changed when applying momentum. Figure 2a shows that Nesterov's momentum hurts the convergence in the first 60 epochs but accelerates the final convergence, which verifies the importance of momentum for achieving high accuracy. Figure 2c suggests that adding Nesterov's momentum slightly increases the uncer-

[5]Tensorflow tracks the values $\{y_k + \beta \cdot v_k\} = \{x_k\}$.

[6]The settings follow Ma and Yarats (2019). Since 90-epoch training is not standard for CIFAR-10, we choose the models that can achieve decent performance in 90 epochs.



(a) Sweeping m in $\{3, 5, 7, 10, 20, 30\}$. Run 5 seeds.

(b) Fixing $m = 5$. Run 20 seeds.

Figure 3: ResNet34 on CIFAR-10. For all methods, $\eta_0 = 0.1, \beta = 0.9$, using same x_0. Labels of AM1-SGD are 'AM1-SGD-{*Option*}'. Shaded bands (or bars) indicate ± 1 standard deviation.

tainty in the training process of SGD.

- *Train-batch loss vs. Full-batch loss.* In Figure 2b, train-batch loss stands for the average of batch losses forwarded in an epoch, which is commonly used to indicate the training process in deep learning. Full-batch loss is the average loss over the entire training dataset evaluated at the end of each epoch. In terms of optimizer evaluation, full-batch loss is much more informative than train-batch loss as it reveals the robustness of an optimizer. However, full-batch loss is expensive to evaluate. On the other hand, test accuracy couples optimization and generalization, but since it is also evaluated at the end of the epoch, its convergence is similar to full-batch loss (see Figure 2a, 2b). Considering the basic usage of momentum in deep learning, we mainly use test accuracy to evaluate optimizers. We provide more discussion on this issue in Appendix C.

- *Robustness.* Inspired by Theorem 1b, we run a method multiple times with different seeds (same x_0) and measure the standard deviation of accuracy or loss at each iterate. Assuming a Gaussian underlying distribution, we can characterize this deviation by $\mathcal{K}_1(m, \Lambda)$ in Theorem 1b with some fixed Λ.

We also plot the convergence of OM-SGD in Figure 2a. Interestingly, OM-SGD performs slightly better in this task: the final accuracies of M-SGD and OM-SGD are $94.61\% \pm 0.15\%$ and $94.73\% \pm 0.11\%$ with average deviations at 1.04% and 0.63%, respectively.

We do not compare with adaptive methods (Duchi et al., 2011; Kingma and Ba, 2015), which scale the gradient using a diagonal matrix to speed up training. Wilson et al. (2017) showed that these methods always generalize poorly compared with SGD with momentum. We chose the tasks where Nesterov's momentum is very effective and popular to conduct our experiments.

Table 1: Detailed data of the curves in Figure 3b.

METHOD	FINAL ACCURACY	Avg. STD
SGD	$93.30\% \pm 0.20\%$	0.93%
M-SGD	$94.71\% \pm 0.17\%$	1.00%
AM1-SGD-I	$94.68\% \pm 0.18\%$	**0.59**%
AM1-SGD-II	$94.62\% \pm 0.15\%$	**0.31**%

6 EXPERIMENTS

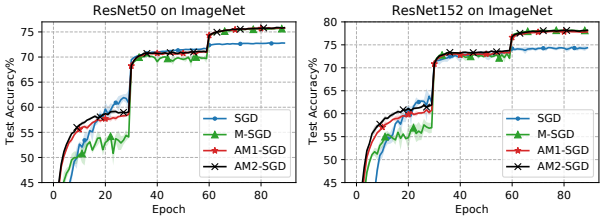
From Theorems 1 and 2, we see that if m is small, the parameters α and β do not change a lot from the case where $m = 1$. This inspired us to align (η, β) of AM1/2-SGD with that of M-SGD and we tune only m. Such a choice facilitates the usage of AM1/2-SGD. In other words, we used the following parameter settings: η for SGD, (η, β) for M-SGD and (η, β, m) for AM1/2-SGD.

6.1 PARAMETER SWEEP ON CIFAR-10

As mentioned in Section 4, the practical effect of amortization is undetermined. Thus, we start with a parameter sweep experiment for m.

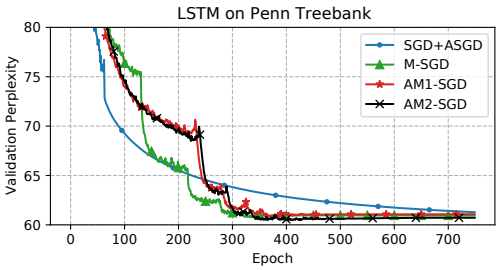
In Figure 3a, we trained ResNet34 on CIFAR-10 using AM1-SGD with various m. The experiments were repeated 5 times with different random seeds to measure the robustness. The convergence behaviors can be found in Appendix A.2. Note that the leftmost points ($m = 1$) in Figure 3a correspond to the results of M-SGD and OM-SGD, which are already given in Figure 2. From this empirical result, we see that m introduces a trade-off between the final accuracy and robustness while the improvement on the robustness is much more significant than the negative effect on the final accuracy. Figure 3a suggests that $m = 5$ is a good choice for this task. For simplicity, and also as a recommended setting, we fix $m = 5$ for the rest of experiments in this paper.

To provide a stronger justification, we ran 20 seeds with $m = 5$ in Figure 3b and the detailed data are given in Ta-



METHOD	ImageNet (Final Accuracy)	
	ResNet50	ResNet152
SGD	$72.78\% \pm 0.08\%$	$74.36\% \pm 0.29\%$
M-SGD	$75.71\% \pm 0.06\%$	$78.07\% \pm 0.10\%$
AM1-SGD	**75.78**$\% \pm 0.11\%$	$77.82\% \pm 0.29\%$
AM2-SGD	**75.85**$\% \pm 0.07\%$	**78.19**$\% \pm 0.15\%$

Figure 4 & Table 2: ResNet on ImageNet. Run 3 seeds. Shaded bands indicate ± 1 standard deviation.



METHOD	Penn Treebank (Perplexity)	
	Validation	Test
SGD+ASGD	61.28	59.07
M-SGD	60.75	58.36
AM1-SGD	60.73	**57.98**
AM2-SGD	**60.43**	58.23

Figure 5 & Table 3: LSTM on Penn Treebank.

ble 1. Recall that Option I omits the tail averaging at the output point. We can thus understand the gap between two options as the effect of tail averaging. Since Option I is basically SGD with the amortized momentum, the results justify that the amortized momentum significantly increases the robustness. It is interesting that the amortized momentum, while being a very large momentum, not only provides acceleration, but also helps the algorithm become more robust than SGD. This observation basically differentiates AM1-SGD from a simple interpolation in-between M-SGD and SGD.

We measured all the wall-clock times in the experiments. However, we observed that even on the same type of GPUs, the running times fluctuate a lot and do not exhibit a clear trend. Roughly speaking, the running time of AM1-SGD ($m = 5$) is improved by $2\% - 3\%$ compared with M-SGD (measured on the same GPU and using the same random batches).

We also did a full-batch loss experiment using a smaller ResNet18 with pre-activation (He et al., 2016b). Since the results resemble Figure 3b, we report them in Appendix A.3.

Learning rate scheduler issue We observed that when we use schedulers with a large decay factor and β is too large for the task (e.g., 0.995 for the task of this section), there would be a performance drop after the learning rate reduction. We believe that it is caused by the different cardinalities of iterates being averaged in $\tilde{x}^+$, which leads to a false momentum. This issue is resolved by restarting the algorithm after each learning rate reduction inspired by (Odonoghue and Candes, 2015). We include more discussion and evidence in Appendix A.6.

6.2 IMAGENET

We trained ResNet50 and ResNet152 (He et al., 2016a) on the ILSVRC2012 dataset ("ImageNet") (Russakovsky et al., 2015) shown in Figure 4. Here we choose to evaluate Option II for AM1/2-SGD, which corresponds to the analysis. From the experiments on CIFAR-10, we see that Option I is basically a "perturbed" version of Option II, while this "perturbation" could lead to a slightly higher final accuracy (see Table 1).

For this task, we used 0.1 initial learning rate and 0.9 momentum for all methods, which is a typical choice. We performed a restart after each learning rate reduction as discussed in Appendix A.6. We believe that this helps the training process and also does not incur any additional overhead. We report the final accuracy in Table 2.

6.3 LANGUAGE MODEL

We did a language model experiment on Penn Treebank dataset (Marcus et al., 1993). We used the LSTM (Hochreiter and Schmidhuber, 1997) model defined in Merity et al. (2018) and followed the experimental setup in its released code. We only changed the learning rate and momentum in the setup. The baseline is SGD+ASGD[7] (Polyak and Juditsky, 1992) with constant learning rate 30 as used in Merity et al. (2018). For the choice of (η, β), following Lucas et al. (2019), we chose $\beta = 0.99$ and used the scheduler that reduces the learning rate by half when the validation loss has not decreased for 15 epochs. We swept η from $\{5, 2.5, 1, 0.1, 0.01\}$ and found that $\eta = 2.5$ resulted in

[7] SGD+ASGD is to run SGD and switch to averaged SGD (ASGD) when a threshold is met.

the lowest validation perplexity for M-SGD. We thus ran AM1-SGD and AM2-SGD (Option II) with this (η, β) and $m = 5$. Due to the small decay factor, we did not restart AM1-SGD and AM2-SGD after learning rate reductions. The convergence of validation perplexity is plotted in Figure 5. We report the lowest validation perplexity and test perplexity in Table 3. This experiment is directly comparable with the one in Lucas et al. (2019).

6.4 A NICE ALTERNATIVE FOR M-SGD

In summary, we list the advantages of AM1-SGD over M-SGD (or from users' angle, the benefits of setting $m > 1$) that are discovered in this section:

(1) Increasing m improves robustness.

(2) Increasing m reduces the (amortized) iteration cost, which is discussed in Appendix A.1.

(3) A suitably chosen m boosts the convergence rate in the early stage of training and produces comparable final generalization performance.

(4) It is easy and safe to tune m. The performance of AM1-SGD are stable for a wide range of m.

Some minor drawbacks of AM1-SGD: it requires one more memory buffer, which is acceptable in most cases, and it shows some undesired behaviors when working with some learning rate schedulers, which can be addressed by performing restarts.

Extra results are provided in the appendices for interested readers: the robustness when using large β in Appendix A.4, a CIFAR-100 experiment in Appendix A.7 and comparison with classical momentum (Polyak, 1964), AggMo (Lucas et al., 2019) and QHM (Ma and Yarats, 2019) in Appendix A.5.

7 CONCLUSIONS

We presented Amortized Nesterov's Momentum, which is a special variant of Nesterov's momentum that utilizes several past iterates. Based on this idea, we designed two different realizations, namely, AM1-SGD and AM2-SGD. We derived optimal convergence rates for them in general convex setting and showed that the effect of amortization is trading off Nesterov's acceleration versus variance control. For deep learning tasks, both of them are simple to implement with little-to-no additional tuning overhead over M-SGD. Our empirical results demonstrate that AM1-SGD can serve as a favorable alternative to M-SGD in large-scale deep learning tasks.

Acknowledgments

We would like to thank the reviewers for their valuable comments and Xiao Yan for his help in revising the paper. This work was partially supported by GRF 14208318 from the RGC and ITF 6904945 from the ITC of HKSAR, and the National Natural Science Foundation of China (NSFC) (Grant No. 61672552).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, pages 265–283.

Allen-Zhu, Z. (2018). Katyusha: The First Direct Acceleration of Stochastic Gradient Methods. *J. Mach. Learn. Res.*, 18(221):1–51.

Auslender, A. and Teboulle, M. (2006). Interior gradient and proximal methods for convex and conic optimization. *SIAM J. Optim.*, 16(3):697–725.

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imag. Sci.*, 2(1):183–202.

Belkin, M., Ma, S., and Mandal, S. (2018). To Understand Deep Learning We Need to Understand Kernel Learning. In *ICML*, pages 541–549.

Ben-Tal, A. and Nemirovski, A. (2013). *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics.

Cyrus, S., Hu, B., Van Scoy, B., and Lessard, L. (2018). A robust accelerated optimization algorithm for strongly convex functions. In *ACC*, pages 1376–1381. IEEE.

Defazio, A. (2019). On the Curved Geometry of Accelerated Optimization. In *NIPS*, pages 1764–1773.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, pages 1646–1654.

Devolder, O., Glineur, F., and Nesterov, Y. (2014). First-order methods of smooth convex optimization with inexact oracle. *Math. Program.*, 146(1-2):37–75.

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

Ghadimi, S. and Lan, G. (2012). Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization i: A generic algorithmic framework. *SIAM J. Optim.*, 22(4):1469–1492.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *CVPR*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hu, C., Pan, W., and Kwok, J. T. (2009). Accelerated gradient methods for stochastic optimization and online learning. In *NIPS*, pages 781–789.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, pages 4700–4708.

Jain, P., Kakade, S. M., Kidambi, R., Netrapalli, P., and Sidford, A. (2018). Accelerating Stochastic Gradient Descent for Least Squares Regression. In *COLT*, pages 545–604.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323.

Kidambi, R., Netrapalli, P., Jain, P., and Kakade, S. M. (2018). On the insufficiency of existing momentum schemes for Stochastic Optimization. In *ICLR*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Lan, G. (2012). An optimal method for stochastic composite optimization. *Math. Program.*, 133(1-2):365–397.

Lan, G., Li, Z., and Zhou, Y. (2019). A unified variance-reduced accelerated gradient method for convex optimization. In *NIPS*, pages 10462–10472.

Lessard, L., Recht, B., and Packard, A. (2016). Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM J. Optim.*, 26(1):57–95.

Liu, C. and Belkin, M. (2020). Accelerating SGD with momentum for over-parameterized learning. In *ICLR*.

Lucas, J., Sun, S., Zemel, R., and Grosse, R. (2019). Aggregated Momentum: Stability Through Passive Damping. In *ICLR*.

Ma, J. and Yarats, D. (2019). Quasi-hyperbolic momentum and Adam for deep learning. In *ICLR*.

Ma, S., Bassily, R., and Belkin, M. (2018). The Power of Interpolation: Understanding the Effectiveness of SGD in Modern Over-parametrized Learning. In *ICML*, pages 3325–3334.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and Optimizing LSTM Language Models. In *ICLR*.

Nazin, A., Nemirovsky, A., Tsybakov, A., and Juditsky, A. (2019). Algorithms of robust stochastic optimization based on mirror descent method. *Autom. Remote. Control.*, 80(9):1607–1627.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM J. Optim.*, 19(4):1574–1609.

Nemirovski, A. and Yudin, D. (1983). *Problem complexity and method efficiency in optimization*. John Wiley, New York.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.

Nesterov, Y. (2013a). Gradient methods for minimizing composite functions. *Math. Program.*, 140(1):125–161.

Nesterov, Y. (2013b). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

Odonoghue, B. and Candes, E. (2015). Adaptive restart for accelerated gradient schemes. *Found. Comput. Math.*, 15(3):715–732.

Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Comput. Math. & Math. Phys.*, 4(5):1–17.

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Stat.*, pages 400–407.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *ICML*, pages 1139–1147.

Tseng, P. (2008). On accelerated proximal gradient methods for convex-concave optimization.

Vaswani, S., Bach, F., and Schmidt, M. (2019). Fast and Faster Convergence of SGD for Over-Parameterized Models and an Accelerated Perceptron. In *AISTATS*, pages 1195–1204.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pages 4148–4158.

Yuan, K., Ying, B., and Sayed, A. H. (2016). On the Influence of Momentum Acceleration on Online Learning. *J. Mach. Learn. Res.*, 17(192):1–66.

Zagoruyko, S. and Komodakis, N. (2016). Wide Residual Networks. In *BMVC*, pages 87.1–87.12.

Zhou, K., Ding, Q., Shang, F., Cheng, J., Li, D., and Luo, Z.-Q. (2019). Direct Acceleration of SAGA using Sampled Negative Momentum. In *AISTATS*, pages 1602–1610.

Zhou, K., Shang, F., and Cheng, J. (2018). A Simple Stochastic Variance Reduced Algorithm with Fast Convergence Rates. In *ICML*, pages 5980–5989.

Supplementary Material for "Amortized Nesterov's Momentum: A Robust Momentum and Its Application to Deep Learning"

A Extra Experimental Results

In this appendix, we provide more experimental results to further evaluate the Amortized Nesterov's Momentum. Table 4 shows the detailed data of the parameter sweep experiments, where the convergence curves of these results are given in Appendix A.2. We discuss our implementations of AM1-SGD and AM2-SGD in Appendix A.1. We report the results of a full-batch loss experiment using ResNet18 in Appendix A.3. In Appendix A.4, we compare the robustness of AM1-SGD and M-SGD on large momentum parameters. In Appendix A.5, we empirically compare the Amortized Nesterov's Momentum with classical momentum (Polyak, 1964), aggregated momentum (Lucas et al., 2019) and quasi-hyperbolic momentum (Ma and Yarats, 2019). We discuss the issues with learning rate schedulers in Appendix A.6. A CIFAR-100 experiment is provided in Appendix A.7. Convex experiments are given in Appendix A.8. We also provide a sanity check for our implementation in Appendix A.9.

Table 4: Final test accuracy and average accuracy STD of training ResNet34 on CIFAR-10 over 5 runs (including the detailed data of the curves in Figure 2 and Figure 3a). For all the methods, $\eta_0 = 0.1, \beta = 0.9$. Multiple runs start with the same x_0.

METHOD	DESCRIPTION	FINAL ACCURACY	Avg. STD
SGD	Standard Pytorch	$93.41\% \pm 0.15\%$	0.99%
M-SGD	Standard Pytorch	$94.61\% \pm 0.15\%$	1.04%
AM1-SGD	Option I, $m = 1$, sanity check	$94.67\% \pm 0.14\%$	0.91%
AM1-SGD	Option I, $m = 3$	$94.63\% \pm 0.03\%$	0.64%
AM1-SGD	Option I, $m = 5$	$94.60\% \pm 0.10\%$	0.50%
AM1-SGD	Option I, $m = 7$	$94.64\% \pm 0.13\%$	0.44%
AM1-SGD	Option I, $m = 10$	$94.54\% \pm 0.13\%$	0.44%
AM1-SGD	Option I, $m = 20$	$94.38\% \pm 0.22\%$	0.40%
AM1-SGD	Option I, $m = 30$	$94.30\% \pm 0.15\%$	0.43%
OM-SGD	AM1-SGD (Opt. II, $m = 1$)	$94.73\% \pm 0.11\%$	0.63%
AM1-SGD	Option II, $m = 3$	$94.66\% \pm 0.14\%$	0.41%
AM1-SGD	Option II, $m = 5$	$94.60\% \pm 0.08\%$	0.27%
AM1-SGD	Option II, $m = 7$	$94.51\% \pm 0.10\%$	0.28%
AM1-SGD	Option II, $m = 10$	$94.42\% \pm 0.12\%$	0.29%
AM1-SGD	Option II, $m = 20$	$94.36\% \pm 0.18\%$	0.31%
AM1-SGD	Option II, $m = 30$	$94.27\% \pm 0.13\%$	0.34%
AM2-SGD	Option I, $m = 1$, sanity check	$94.68\% \pm 0.21\%$	0.82%
AM2-SGD	Option I, $m = 5$	$94.57\% \pm 0.19\%$	0.59%
AM2-SGD	Option I, $m = 10$	$94.44\% \pm 0.14\%$	0.74%
AM2-SGD	Option I, $m = 20$	$94.31\% \pm 0.15\%$	0.74%
AM2-SGD	Option II, $m = 5$	$94.66\% \pm 0.11\%$	0.26%
AM2-SGD	Option II, $m = 10$	$94.50\% \pm 0.21\%$	0.28%
AM2-SGD	Option II, $m = 20$	$94.41\% \pm 0.14\%$	0.25%

A.1 Implementing AM1-SGD and AM2-SGD

Similar to M-SGD, it is easier to implement Option I in existing deep learning frameworks. To implement Option II, we can either maintain another identical network for the shifted point ($\tilde{x}$ or $\tilde{\phi}$) or temporarily change the network parameters in the evaluation phase. In our implementations of AM1-SGD and AM2-SGD, we simply adopt the latter solution.

Alg. 5 AM1-SGD (Alg. 2 with improved efficiency)

Input: Initial guess x_0, learning rate η, momentum β, amortization length m, iteration number K.
Initialize: $x \leftarrow x_0, \tilde{x} \leftarrow x_0, \tilde{v}^+ \leftarrow -m \cdot x_0$.

for $k = 0, \ldots, K-1$ **do**
  $x \leftarrow x - \eta \cdot \nabla f_{i_k}(x)$.
  $\tilde{v}^+ \leftarrow \tilde{v}^+ + x$.
  if $(k+1) \mod m = 0$ **then**
    $x \leftarrow x + (\beta/m) \cdot \tilde{v}^+$.
    $\tilde{x} \leftarrow \tilde{x} + (1/m) \cdot \tilde{v}^+, \tilde{v}^+ \leftarrow -m \cdot \tilde{x}$.
  end if
end for

Output: Option I: x, Option II: $\tilde{x}$.

For AM1-SGD, we can improve the efficiency of Algorithm 2 by maintaining a running scaled momentum $\tilde{v}^+ \triangleq m \cdot (\tilde{x}^+ - \tilde{x})$ instead of the running average $\tilde{x}^+$ as shown in Algorithm 5. Then, in one m-iterations loop, for each of the first $m-1$ iterations, AM1-SGD requires 1 vector addition and 1 scaled vector addition. At the m-th iteration, it requires 1 vector addition, 1 scalar-vector multiplication and 3 scaled vector additions. In comparison, M-SGD (standard PyTorch) requires 1 vector addition, 1 (in-place) scalar-vector multiplication and 2 scaled vector additions per iteration. Thus, as long as $m > 2$, AM1-SGD has lower amortized cost than M-SGD. In terms of memory complexity, AM1-SGD requires one more auxiliary buffer than M-SGD. For AM2-SGD, we implement Algorithm 4. Note that at each iteration, we sample an index in $[m]$ as j_{k+1} and obtain the stored index j_k.

A.2 The effect of m on convergence

We show in Figure 6 how m affects the convergence of test accuracy. The results show that increasing m speeds up the convergence in the early stage. While for AM1-SGD the convergences of Option I and Option II are similar, AM2-SGD with Option II is consistently better than with Option I in this experiment. It seems that AM2-SGD with Option I does not benefit from increasing m and the algorithm is not robust. Thus, we do not recommend using Option I for AM2-SGD.

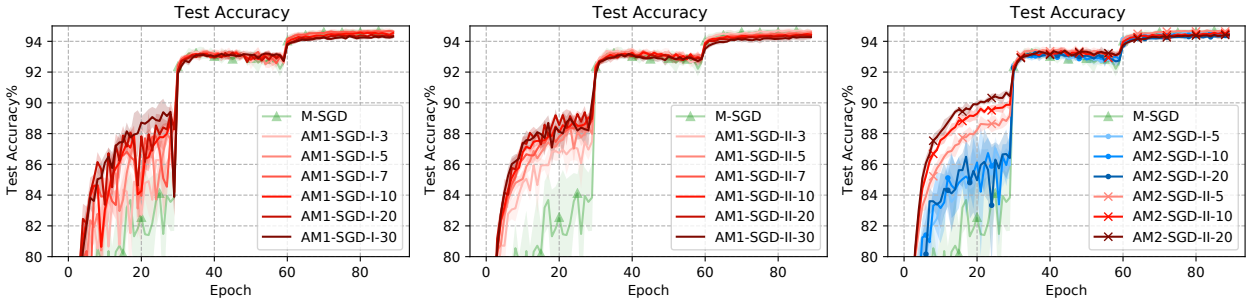
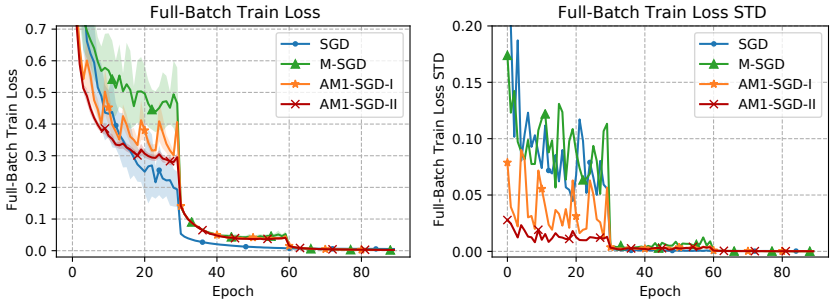


Figure 6: Convergence of test accuracy from the parameter sweep experiments in Table 4. Labels are formatted as 'AM1/2-SGD-{*Option*}-{m}'.

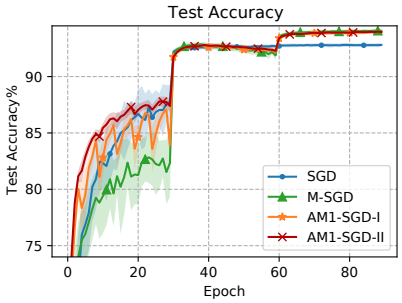
A.3 Full-batch loss experiment

We did a full-batch loss experiment of training ResNet18 with pre-activation (He et al., 2016b) on CIFAR-10 in Figure 7 & Table 5. The accuracy results are given in Figure 8 & Table 6. These results are reminiscent of the ResNet34 experiments (Figure 3b and Table 1).



METHOD	Avg. STD
SGD	0.033
M-SGD	0.034
AM1-SGD-I	**0.014**
AM1-SGD-II	**0.006**

Figure 7 & Table 5: ResNet18 with pre-activation on CIFAR-10. For all methods, $\eta_0 = 0.1, \beta = 0.9$, run 20 seeds. For AM1-SGD, $m = 5$ and its labels are formatted as 'AM1-SGD-{*Option*}'. Shaded bands indicate ± 1 standard deviation.

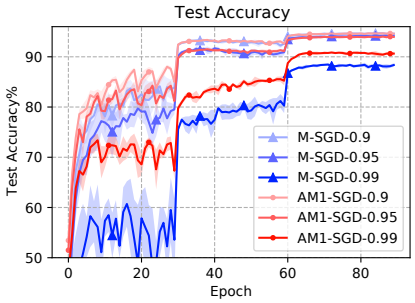


METHOD	FINAL ACCURACY	Avg. STD
SGD	$92.81\% \pm 0.15\%$	1.01%
M-SGD	$94.06\% \pm 0.17\%$	1.10%
AM1-SGD-I	$93.97\% \pm 0.15\%$	**0.54**%
AM1-SGD-II	$93.95\% \pm 0.19\%$	**0.32**%

Figure 8 & Table 6: ResNet18 with pre-activation on CIFAR-10. For all methods, $\eta_0 = 0.1, \beta = 0.9$, run 20 seeds. For AM1-SGD, $m = 5$. Shaded bands indicate ± 1 standard deviation.

A.4 Robustness on large momentum parameters

We compare the robustness of M-SGD and AM1-SGD when β is large in Figure 9 & Table 7. AM1-SGD uses Option I, which omits the tail averaging effect at the output point. As we can see, the STD error of M-SGD scales up significantly when β is larger and the performance is more affected by a large β compared with AM1-SGD.



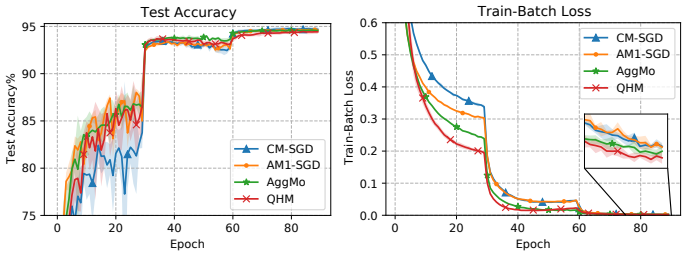
METHOD	FINAL ACCURACY	Avg. STD
M-SGD-0.9	$94.61\% \pm 0.15\%$	1.04%
M-SGD-0.95	$94.20\% \pm 0.12\%$	1.20%
M-SGD-0.99	$88.37\% \pm 0.36\%$	2.56%
AM1-SGD-0.9	$94.60\% \pm 0.10\%$	0.50%
AM1-SGD-0.95	$93.94\% \pm 0.07\%$	0.58%
AM1-SGD-0.99	$90.64\% \pm 0.38\%$	0.90%

Figure 9 & Table 7: ResNet34 on CIFAR-10. $\eta_0 = 0.1, \beta \in \{0.9, 0.95, 0.99\}$, run 5 seeds (the $\beta = 0.9$ results are copied from Table 4). Labels are formatted as "{*Algorithm*}-{β}".

A.5 Comparison with other momentum

In this section, we compare AM1-SGD (Option I) with classical momentum (Polyak, 1964), AggMo (Lucas et al., 2019) and QHM (Ma and Yarats, 2019) in our basic case study (training ResNet34 on CIFAR-10). Since we are not

aware of what makes a fair comparison with these methods (e.g., it is not clear what is the effective learning rate for AM1-SGD), we compare them based on the default hyper-parameter settings suggested by their papers.



METHOD	FINAL ACCURACY	Avg. STD
CM-SGD	$94.69\% \pm 0.21\%$	1.11%
AM1-SGD	$94.60\% \pm 0.10\%$	0.50%
AggMo	$94.56\% \pm 0.14\%$	0.78%
QHM	$94.43\% \pm 0.23\%$	1.07%
M-SGD	$94.61\% \pm 0.15\%$	1.04%

Figure 10 & Table 8: ResNet34 on CIFAR-10. Run 5 seeds. The results of AM1-SGD and M-SGD are copied from Table 4.

Classical Momentum The SGD with classical momentum (CM-SGD) that is widely used in deep learning has the following scheme (standard PyTorch) ($v^{cm} \in \mathbb{R}^d, v_0^{cm} = \mathbf{0}$):

$$v_{k+1}^{cm} = \beta \cdot v_k^{cm} + \nabla f_{i_k}(x_k),$$
$$x_{k+1} = x_k - \eta \cdot v_{k+1}^{cm}, \text{ for } k \geq 0.$$

CM-SGD with its typical hyper-parameter settings ($\eta_0 = 0.1, \beta = 0.9$) is observed to achieve similar generalization performance as M-SGD. However, CM-SGD is more unstable and prone to oscillations (Lucas et al., 2019), which makes it less robust than M-SGD as shown in Table 8.

Aggregated Momentum (AggMo) AggMo combines multiple momentum buffers, which is inspired by the passive damping from physics literature (Lucas et al., 2019). AggMo uses the following update rules (for $t = 1, \ldots, T$, $v^{(t)} \in \mathbb{R}^d, v_0^{(t)} = \mathbf{0}$):

$$v_{k+1}^{(t)} = \beta^{(t)} \cdot v_k^{(t)} - \nabla f_{i_k}(x_k), \text{ for } t = 1, \ldots, T,$$
$$x_{k+1} = x_k + \frac{\eta}{T} \cdot \sum_{t=1}^{T} v_{k+1}^{(t)}, \text{ for } k \geq 0.$$

We used the exponential hyper-parameter setting recommended in the original work with the scale-factor $a = 0.1$ fixed, $\beta^{(t)} = 1 - a^{t-1}$, for $t = 1, \ldots, T$ and choosing T in $\{2, 3, 4\}$. We found that $T = 2$ gave the best performance in this experiment. As shown in Figure 10 & Table 8, with the help of passive damping, AggMo is more stable and robust compared with CM-SGD.

Quasi-hyperbolic Momentum (QHM) Ma and Yarats (2019) introduce the immediate discount factor $\nu \in \mathbb{R}$ for the momentum scheme, which results in the QHM update rules ($\alpha \in \mathbb{R}, v^{qh} \in \mathbb{R}^d, v_0^{qh} = \mathbf{0}$):

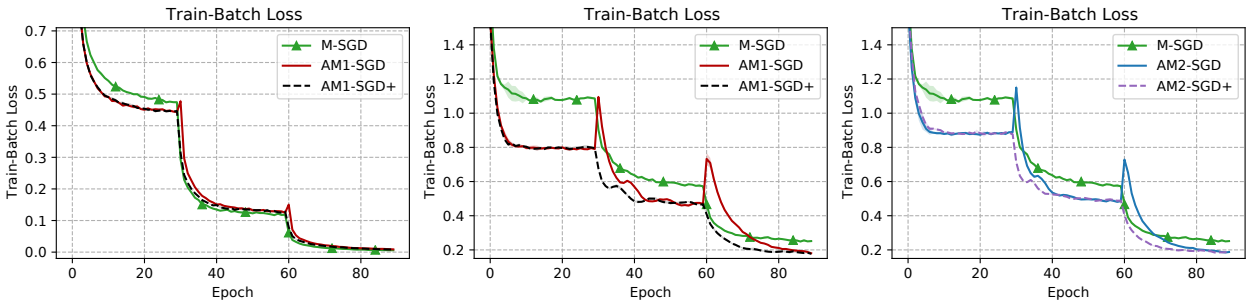
$$v_{k+1}^{qh} = \beta \cdot v_k^{qh} + (1 - \beta) \cdot \nabla f_{i_k}(x_k),$$
$$x_{k+1} = x_k - \alpha \cdot (\nu \cdot v_{k+1}^{qh} + (1 - \nu) \cdot \nabla f_{i_k}(x_k)), \text{ for } k \geq 0.$$

Here we used the recommended hyper-parameter setting for QHM ($\alpha_0 = 1.0, \beta = 0.999, \nu = 0.7$).

Figure 10 shows that AM1-SGD, AggMo and QHM achieve faster convergence in the early stage while CM-SGD has the highest final accuracy. In terms of robustness, huge gaps are observed when comparing AM1-SGD with the remaining methods in Table 8. Note that AM1-SGD is more efficient than both QHM and AggMo, and is as efficient as CM-SGD.

We also plot the convergence of train-batch loss for all the methods in Figure 10. Despite of showing worse generalization performance, both QHM and AggMo perform better on reducing the train-batch loss in this experiment, which is consistent with the results reported in Ma and Yarats (2019); Lucas et al. (2019).

A.6 Issues with learning rate schedulers



(a) $\beta = 0.95$ (b) $\beta = 0.995$ (c) $\beta = 0.995$

Figure 11: ResNet18 on CIFAR-10. $\eta_0 = 0.1, \beta \in \{0.95, 0.995\}$. '+' represents performing a restart after each learning rate reduction.

We show in Figure 11 that when β is large for the task, using step learning rate scheduler with decay factor 10, a performance drop is observed after each reduction. We fix this issue by performing a restart after each learning rate reduction (labeled with '+'). We plot the train-batch loss here because we find that the phenomenon is clearer in this way. Note that output options do not affect the convergence of train-batch loss, and thus this phenomenon exists for both options. If $\beta = 0.9$, there is no observable performance drop in this experiment.

For smooth-changing schedulers such as the cosine annealing scheduler (Loshchilov and Hutter, 2017), the amortized momentum works well as shown in Figure 12.

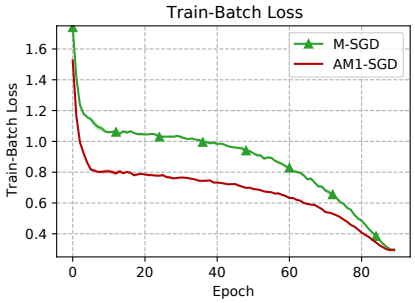


Figure 12: ResNet18 on CIFAR-10. Cosine annealing scheduler (without restarts), $\eta_0 = 0.1, \beta = 0.995$.

A.7 CIFAR-100 experiment

We report the results of training DenseNet121 (Huang et al., 2017) on CIFAR-100 in Figure 13, which shows that both AM1-SGD and AM2-SGD perform well before the final learning rate reduction. However, the final accuracies are lowered around 0.6% compared with M-SGD. We also notice that SGD reduces the train-batch loss at an incredibly fast rate and the losses it reaches are consistently lower than other methods in the entire 300 epochs. However, this performance is not reflected in the convergence of test accuracy. We believe that this phenomenon suggests that the DenseNet model is actually "overfitting" M-SGD (since in the ResNet experiments, M-SGD always achieves a lower train loss than SGD after the final learning rate reduction).

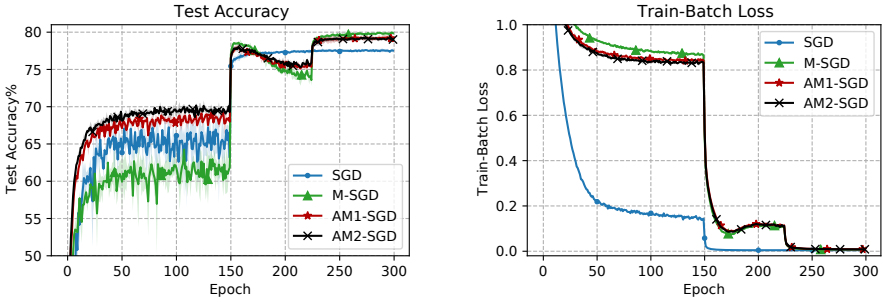


Figure 13: DenseNet121 on CIFAR-100. For all methods, $\eta_0 = 0.1, \beta = 0.9$, run 3 seeds. AM1-SGD and AM2-SGD use Option II and $m = 5$. Shaded bands indicate ± 1 standard deviation.

A.8 Convex experiments

We provide empirical results for AM1-SGD (Algorithm 1) to justify its convergence guarantee (Theorem 1). We consider the following simple convex empirical risk minimization task:

$$\text{Logistic Regression: } f(x) = \frac{1}{|D|}\sum_{i=1}^{|D|} \log\left(1+\exp\left(-b_i\left\langle a_i, x\right\rangle\right)\right), x \in \mathbb{R}^n, a_i \in \mathbb{R}^n, b_i \in \{-1,+1\}, \forall i \in [|D|].$$

We used the a5a dataset ($|D| = 6414$, $n = 123$) from LIBSVM (Chang and Lin, 2011). By normalizing the dataset, we have $L = 0.25, M = 0$ in Assumption (a). The stochastic gradient oracle is defined as $\nabla f_i(x) = \nabla f(x) + \delta$, where δ is sampled uniformly from the sphere $\mathbb{B}(\mathbf{0}, \sigma)$ (which is centered at $\mathbf{0}$ and its radius is σ). This oracle satisfies Assumptions (b) and (c) (and also the "light tail" assumption). This type of noise is frequently used to escape saddle points in non-convex optimization or to ensure differential privacy. We fixed $x_0 = \mathbf{0}$ and estimated $V_d(x^\star, x_0) = \frac{1}{2}\|x_0 - x^\star\|_2^2$ as 350 by running a small amount of iterates. Then we can run AM1-SGD with the parameter choices specified in Theorem 1.

We compared AM1-SGD with M-SGD (or AC-SA, which corresponds to choosing $m = 1$). In Theorem 1, when σ is large, AM1-SGD is expected to converge faster than M-SGD and it also has a smaller deviation predicted by Theorem 1b. We justify these predictions in Figure 14.

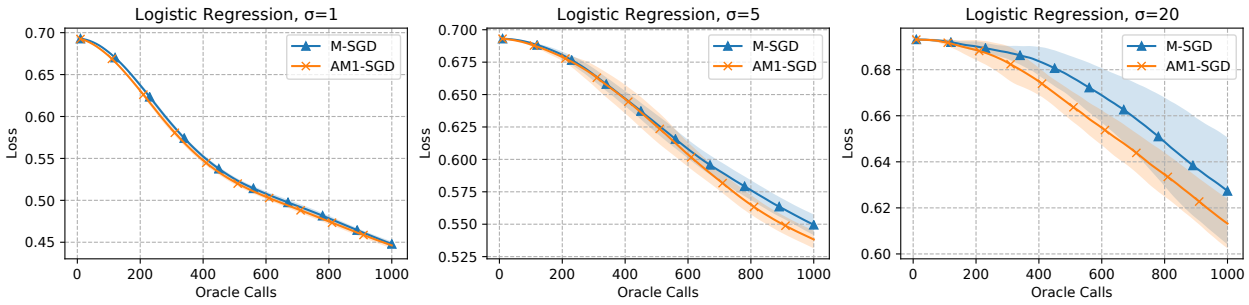


Figure 14: Comparisons between AM1-SGD and M-SGD (AC-SA) in different noise levels. $m = 10$, run 10 seeds. Shaded bands indicate ± 1 standard deviation.

We also studied the effect of choosing different m in Figure 15. The results are similar to the deep learning experiment in Figure 3a (right): m needs to be sufficiently large to show the effect, and after some point, not much benefit can be obtained by further increasing m.

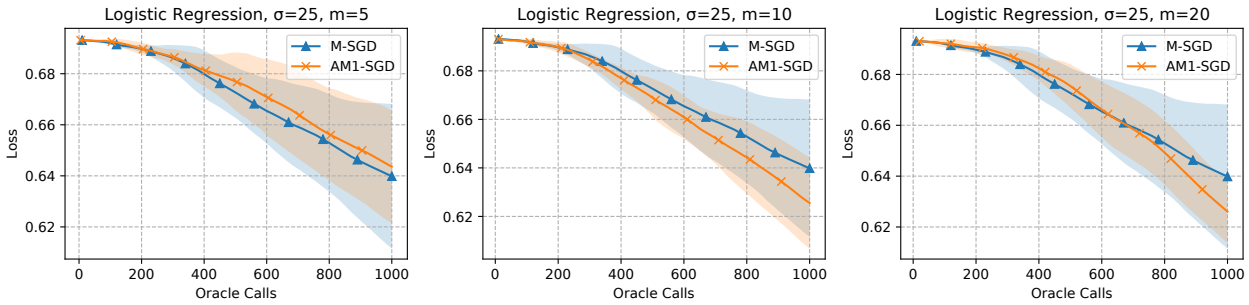


Figure 15: Effect of choosing different m. Run 10 seeds. Shaded bands indicate ± 1 standard deviation.

A.9 Sanity check

When $m = 1$, both AM1-SGD and AM2-SGD (Option I) are equivalent to M-SGD, we plot their convergence in Figure 16 as a sanity check (the detailed data is given in Table 4).

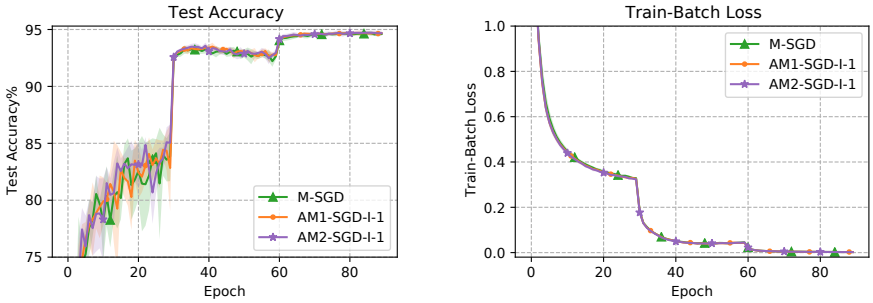


Figure 16: A sanity check. Labels are formatted as 'AM{1/2}-SGD-{*Option*}-{m}'.

We observed that when $m = 1$, both AM1-SGD and AM2-SGD have a lower STD error than M-SGD. We believe that it is because they both maintain the iterates without scaling, which is numerically more stable than M-SGD (M-SGD in standard PyTorch maintains a scaled buffer, i.e., $v_k^{pt} = \eta^{-1}\beta^{-1} \cdot (y_k - x_k)$).

B Missing parts in Section 3 and Section 4

B.1 The reformulations

When in the Euclidean setting, $h \equiv 0$ and β is a constant, we do the reformulations by eliminating the sequence $\{z_k\}$. In this case, the general scheme (2) can be written as

$$\begin{aligned} x &= (1-\beta)\cdot z + \beta \cdot y, \\ z^+ &= z - \alpha \cdot \nabla f_i(x), \\ y^+ &= (1-\beta)\cdot z^+ + \beta \cdot y. \end{aligned}$$

Note that

$$\begin{aligned} y^+ &= (1-\beta)\cdot (z - \alpha \cdot \nabla f_i(x)) + \beta \cdot y \\ &= x - \alpha(1-\beta)\cdot \nabla f_i(x). \end{aligned} \tag{7}$$

For AM1-SGD (Algorithm 1), based on (7), we see that the inner loops are basically SGD,

$$\begin{aligned} x_k &= (1-\beta)\cdot z_k + \beta \cdot \tilde{x}_s, \\ z_{k+1} &= z_k - \alpha \cdot \nabla f_{i_k}(x_k), \\ (x_{k+1} &= (1-\beta)\cdot z_{k+1} + \beta \cdot \tilde{x}_s.) \end{aligned} \quad \begin{array}{c} \alpha(1-\beta)=\eta \\ \text{Eliminate } \{z_k\} \\ \Longrightarrow \end{array} \quad x_{k+1} = x_k - \eta \cdot \nabla f_{i_k}(x_k).$$

At the end of each inner loop (i.e., when $(k+1) \mod m = 0$), we have

$$x_{(s+1)m} = (1-\beta)\cdot z_{(s+1)m} + \beta \cdot \tilde{x}_s,$$

while at the beginning of the next inner loop,

$$x_{(s+1)m} = (1-\beta)\cdot z_{(s+1)m} + \beta \cdot \tilde{x}_{s+1},$$

which means that we need to set $x_{k+1} \leftarrow x_{k+1} + \beta \cdot (\tilde{x}_{s+1} - \tilde{x}_s)$ (reassign the value of x_{k+1}).

For AM2-SGD (Algorithm 3), based on (7),

$$\begin{aligned} x_k^{j_k} &= (1-\beta)\cdot z_k + \beta \cdot \phi_{j_k}^k, \\ z_{k+1} &= z_k - \alpha \cdot \nabla f_{i_k}(x_k^{j_k}), \\ \phi_{j_k}^{k+1} &= (1-\beta)\cdot z_{k+1} + \beta \cdot \phi_{j_k}^k, \\ \big(x_{k+1}^{j_{k+1}} &= (1-\beta)\cdot z_{k+1} + \beta \cdot \phi_{j_{k+1}}^{k+1}\big). \end{aligned} \quad \begin{array}{c} \alpha(1-\beta)=\eta \\ \text{Eliminate } \{z_k\} \\ \Longrightarrow \end{array} \quad \begin{aligned} \phi_{j_k}^{k+1} &= x_k^{j_k} - \eta \cdot \nabla f_{i_k}(x_k^{j_k}), \\ x_{k+1}^{j_{k+1}} &= \phi_{j_k}^{k+1} + \beta \cdot \left(\phi_{j_{k+1}}^{k+1} - \phi_{j_k}^k\right). \end{aligned}$$

AM2-SGD **Algorithm 4**

We also give the reformulation between M-SGD (scheme (4)) and AC-SA (Lan, 2012) for reference:

$$\begin{aligned} x_k &= (1-\beta)\cdot z_k + \beta \cdot y_k, \\ z_{k+1} &= z_k - \alpha \cdot \nabla f_{i_k}(x_k), \\ y_{k+1} &= (1-\beta)\cdot z_{k+1} + \beta \cdot y_k, \\ \big(x_{k+1} &= (1-\beta)\cdot z_{k+1} + \beta \cdot y_{k+1}\big). \end{aligned} \quad \begin{array}{c} \alpha(1-\beta)=\eta \\ \text{Eliminate } \{z_k\} \\ \Longrightarrow \end{array} \quad \begin{aligned} y_{k+1} &= x_k - \eta \cdot \nabla f_{i_k}(x_k), \\ x_{k+1} &= y_{k+1} + \beta \cdot (y_{k+1} - y_k). \end{aligned}$$

AC-SA (Lan, 2012)
(Auslender and Teboulle (2006))

M-SGD (Nesterov, 1983, 2013)

Intuition for the Auslender and Teboulle (2006) scheme can be found in Remark 2 in Lan (2012).

B.2 Proofs of Theorem 1 and Theorem 2

B.2.1 Proof of Lemma 1

This Lemma is provided in a similar way as in Lan (2012); Ghadimi and Lan (2012), we give a proof here for completeness.

Based on the convexity (Assumption (a)), we have

$$\begin{aligned} f(x) - f(x^\star) \leq & \underbrace{\langle \nabla f(x), x - z \rangle}_{R_0} + \underbrace{\langle \nabla f(x) - \nabla f_i(x), z - x^\star \rangle}_{R_1} + \underbrace{\langle \nabla f_i(x), z - z^+ \rangle}_{R_2} \\ & + \underbrace{\langle \nabla f_i(x), z^+ - x^\star \rangle}_{R_3}. \end{aligned} \tag{8}$$

We upper bound the terms on the right side one-by-one.

For R_0,

$$R_0 \overset{(\star)}{=} \frac{\beta}{1-\beta} \langle \nabla f(x), y - x \rangle \leq \frac{\beta}{1-\beta} \big(f(y) - f(x) \big), \tag{9}$$

where $(\star)$ uses the relation between x and z, i.e., $(1-\beta) \cdot (x - z) = \beta \cdot (y - x)$.

For R_2, based on Assumption (a), we have

$$f(y^+) - f(x) + \langle \nabla f(x), x - y^+ \rangle \leq \frac{L}{2} \left\| x - y^+ \right\|^2 + M \left\| x - y^+ \right\|.$$

Then, noting that $x - y^+ = (1-\beta) \cdot (z - z^+)$, we can arrange the above inequality as

$$\begin{aligned} R_2 \leq & \frac{L(1-\beta)}{2} \left\| z - z^+ \right\|^2 + \frac{1}{1-\beta} \big(f(x) - f(y^+) \big) + \langle \nabla f(x) - \nabla f_i(x), z^+ - z \rangle \\ & + M \left\| z - z^+ \right\| \\ \leq & \frac{L(1-\beta)}{2} \left\| z - z^+ \right\|^2 + \frac{1}{1-\beta} \big(f(x) - f(y^+) \big) + \big(\left\| \nabla f(x) - \nabla f_i(x) \right\|_* + M \big) \left\| z - z^+ \right\|. \end{aligned}$$

Using Young's inequality with $\zeta > 0$, we obtain

$$R_2 \leq \frac{L(1-\beta) + \zeta}{2} \left\| z - z^+ \right\|^2 + \frac{1}{1-\beta} \big(f(x) - f(y^+) \big) + \frac{(\left\| \nabla f(x) - \nabla f_i(x) \right\|_* + M)^2}{2\zeta}. \tag{10}$$

For R_3, based on the optimality condition of

$$\mathrm{Prox}_{\alpha h}\big(z, \alpha \cdot \nabla f_i(x) \big) \triangleq \arg\min_{u \in X} \big\{ V_d(u, z) + \alpha \langle \nabla f_i(x), u \rangle + \alpha h(u) \big\},$$

and denoting $\partial h(z^+) \in E^*$ as a subgradient of h at z^+, we have for any $w \in X$,

$$\begin{aligned} & \big\langle \nabla d(z^+) - \nabla d(z) + \alpha \cdot \nabla f_i(x) + \alpha \cdot \partial h(z^+), w - z^+ \big\rangle \geq 0, \\ \big\langle \nabla f_i(x), z^+ - w \big\rangle \leq & \big\langle \partial h(z^+), w - z^+ \big\rangle + \frac{1}{\alpha} \big\langle \nabla d(z^+) - \nabla d(z), w - z^+ \big\rangle \\ \leq & h(w) - h(z^+) + \frac{1}{\alpha} \big\langle \nabla d(z^+) - \nabla d(z), w - z^+ \big\rangle. \end{aligned}$$

Choosing $w = x^\star$ and applying the triangle equality of Bregman divergence, we obtain

$$\begin{aligned} R_3 \leq & h(x^\star) - h(z^+) + \frac{1}{\alpha} \big\langle \nabla d(z^+) - \nabla d(z), x^\star - z^+ \big\rangle \\ = & h(x^\star) - h(z^+) + \frac{1}{\alpha} \big(V_d(x^\star, z) - V_d(x^\star, z^+) - V_d(z^+, z) \big) \\ \overset{(\star)}{\leq} & h(x^\star) - h(z^+) + \frac{1}{\alpha} \big(V_d(x^\star, z) - V_d(x^\star, z^+) \big) - \frac{1}{2\alpha} \left\| z^+ - z \right\|^2, \end{aligned} \tag{11}$$

where $(\star)$ follows from the 1-strong convexity of d, which implies that $V_d(x,y) \geq \frac{1}{2}\|x-y\|^2, \forall x,y \in X$. Finally, by upper bounding (8) using (9), (10), (11), we conclude that

$$\begin{aligned} f(x) - f(x^\star) \leq R_1 &+ \frac{\beta}{1-\beta}\big(f(y) - f(x)\big) + \frac{L(1-\beta)+\zeta-\alpha^{-1}}{2}\left\|z - z^+\right\|^2 \\ &+ \frac{1}{1-\beta}\big(f(x) - f(y^+)\big) + h(x^\star) - h(z^+) + \frac{(\|\nabla f(x) - \nabla f_i(x)\|_* + M)^2}{2\zeta} \\ &+ \frac{1}{\alpha}\left(V_d(x^\star, z) - V_d(x^\star, z^+)\right), \end{aligned}$$

After simplification,

$$\begin{aligned} \frac{1}{1-\beta}\big(f(y^+) - f(x^\star)\big) \leq \frac{\beta}{1-\beta}\big(f(y) - f(x^\star)\big) &+ \frac{L(1-\beta)+\zeta-\alpha^{-1}}{2}\left\|z - z^+\right\|^2 \\ &+ h(x^\star) - h(z^+) + \frac{(\|\nabla f(x) - \nabla f_i(x)\|_* + M)^2}{2\zeta} + R_1 \\ &+ \frac{1}{\alpha}\left(V_d(x^\star, z) - V_d(x^\star, z^+)\right). \end{aligned} \tag{12}$$

Note that with the convexity of h and $y^+ = (1-\beta)\cdot z^+ + \beta \cdot y$, we have

$$\begin{aligned} h(y^+) &\leq (1-\beta)h(z^+) + \beta h(y), \\ h(z^+) &\geq \frac{1}{1-\beta}h(y^+) - \frac{\beta}{1-\beta}h(y). \end{aligned}$$

Using the above inequality and choosing $\zeta = \alpha^{-1} - L(1-\beta) > 0 \Rightarrow \alpha(1-\beta) < 1/L$, we can arrange (12) as

$$\begin{aligned} \frac{1}{1-\beta}\big(F(y^+) - F(x^\star)\big) \leq \frac{\beta}{1-\beta}\big(F(y) - F(x^\star)\big) &+ \frac{1}{\alpha}\left(V_d(x^\star, z) - V_d(x^\star, z^+)\right) \\ &+ \frac{(\|\nabla f(x) - \nabla f_i(x)\|_* + M)^2}{2(\alpha^{-1} - L(1-\beta))} + R_1. \end{aligned}$$

B.2.2 Proof of Theorem 1a

Using Assumption (c), Lemma 1 with

$$\begin{cases} x = x_k \\ z = z_k \\ z^+ = z_{k+1} \\ y = \tilde{x}_s \\ y^+ = x_{k+1} \\ \alpha = \alpha_s \\ \beta = \beta_s \end{cases}, \tag{13}$$

and taking expectation, if $\alpha_s(1-\beta_s) < 1/L$, we have

$$\begin{aligned} &\frac{1}{1-\beta_s}\big(\mathbb{E}_{i_k}\left[F(x_{k+1})\right] - F(x^\star)\big) + \frac{1}{\alpha_s}\mathbb{E}_{i_k}\left[V_d(x^\star, z_{k+1})\right] \\ \leq &\frac{\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{\alpha_s}V_d(x^\star, z_k) + \frac{(\sigma+M)^2}{2(\alpha_s^{-1} - L(1-\beta_s))}. \end{aligned}$$

Summing the above inequality from $k = sm, \ldots, sm + m - 1$, we obtain

$$\frac{1}{(1-\beta_s)m}\sum_{j=1}^{m}\big(\mathbb{E}\left[F(x_{sm+j})\right] - F(x^\star)\big) + \frac{1}{\alpha_s m}\mathbb{E}\left[V_d(x^\star, z_{(s+1)m})\right]$$
$$\leq \frac{\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{\alpha_s m}V_d(x^\star, z_{sm}) + \frac{(\sigma+M)^2}{2(\alpha_s^{-1} - L(1-\beta_s))},$$

Using the definition of $\tilde{x}_{s+1}$ and convexity,

$$\begin{aligned}&\frac{\alpha_s}{1-\beta_s}\big(\mathbb{E}\left[F(\tilde{x}_{s+1})\right] - F(x^\star)\big) + \frac{1}{m}\mathbb{E}\left[V_d(x^\star, z_{(s+1)m})\right]\\ &\leq \frac{\alpha_s\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{m}V_d(x^\star, z_{sm}) + \frac{\alpha_s(\sigma^2+M^2)}{\alpha_s^{-1} - L(1-\beta_s)}.\end{aligned} \tag{14}$$

It can be verified that with the choices $\beta_s = \frac{s}{s+2}$ and $\alpha_s = \frac{\lambda_1}{L(1-\beta_s)}$, the following holds for $s \geq 0$,

$$\frac{\alpha_{s+1}\beta_{s+1}}{1-\beta_{s+1}} \leq \frac{\alpha_s}{1-\beta_s} \text{ and } \beta_0 = 0. \tag{15}$$

Thus, by telescoping (14) from $s = S-1, \ldots, 0$, we obtain

$$\frac{\alpha_{S-1}}{1-\beta_{S-1}}\big(\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star)\big) + \frac{1}{m}\mathbb{E}\left[V_d(x^\star, z_{Sm})\right]$$
$$\leq \frac{1}{m}V_d(x^\star, x_0) + \sum_{s=0}^{S-1}\frac{\alpha_s(\sigma^2+M^2)}{\alpha_s^{-1} - L(1-\beta_s)},$$

and thus,

$$\begin{aligned}\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star) &\leq \frac{4L}{\lambda_1 m(S+1)^2}V_d(x^\star, x_0) + \frac{4L(\sigma^2+M^2)}{\lambda_1(S+1)^2}\sum_{s=0}^{S-1}\frac{\alpha_s^2}{1-\alpha_s(1-\beta_s)L}\\ &\overset{(a)}{\leq} \frac{4L}{\lambda_1 m(S+1)^2}V_d(x^\star, x_0) + \frac{3\lambda_1(\sigma^2+M^2)}{L(S+1)^2}\sum_{s=0}^{S-1}(s+2)^2\\ &\overset{(b)}{\leq} \frac{4L}{\lambda_1 m(S+1)^2}V_d(x^\star, x_0) + \frac{8\lambda_1(\sigma^2+M^2)(S+1)}{L},\end{aligned}$$

where (a) follows from $\lambda_1 \leq \frac{2}{3}$ and (b) holds because $0 \leq x \mapsto (x+2)^2$ is non-decreasing and thus

$$\sum_{s=0}^{S-1}(s+2)^2 \leq \int_0^S (x+2)^2 dx \leq \frac{(S+2)^3}{3} \leq \frac{8(S+1)^3}{3}.$$

Denoting

$$\lambda_1^\star \triangleq \frac{L\sqrt{V_d(x^\star, x_0)}}{\sqrt{2m}\sqrt{\sigma^2+M^2}(S+1)^{\frac{3}{2}}},$$

and based on the choice of $\lambda_1 = \min\left\{\frac{2}{3}, \lambda_1^*\right\}$, if $\lambda_1^* \leq \frac{2}{3}$, we have

$$\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star) \leq \frac{8\sqrt{2V_d(x^\star, x_0)}\sqrt{\sigma^2+M^2}}{m^{\frac{1}{2}}(S+1)^{\frac{1}{2}}}.$$

If $\lambda_1^* > \frac{2}{3}$,

$$\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star) \leq \frac{6LV_d(x^\star, x_0)}{m(S+1)^2} + \frac{4\sqrt{2V_d(x^\star, x_0)}\sqrt{\sigma^2+M^2}}{m^{\frac{1}{2}}(S+1)^{\frac{1}{2}}}.$$

Thus, we conclude that

$$\mathbb{E}\left[F(\tilde{x}_S)\right] - F(x^\star) \leq \frac{6LV_d(x^\star, x_0)}{m(S+1)^2} + \frac{8\sqrt{2V_d(x^\star, x_0)}\sqrt{\sigma^2 + M^2}}{m^{\frac{1}{2}}(S+1)^{\frac{1}{2}}}.$$

Substituting $S = K/m$ completes the proof.

B.2.3 Proof of Theorem 1b

In order to prove Theorem 1b, we need the following known result for the martingale difference (cf. Lemma 2 in Lan et al. (2012)):

Lemma 2. *With $N > 0$, let $\xi_0, \xi_1, \ldots, \xi_{N-1}$ be a sequence of i.i.d. random variables, for $t = 0, \ldots, N-1$, $\sigma_t > 0$ be a deterministic number and $\psi_t = \psi_t(\xi_0, \ldots, \xi_t)$ be a deterministic measurable function such that $\mathbb{E}_{\xi_t}[\psi_t] = 0$ a.s. and $\mathbb{E}_{\xi_t}\left[\exp\{\psi_t^2/\sigma_t^2\}\right] \leq \exp\{1\}$ a.s.. Then for any $\Lambda \geq 0$,*

$$\text{Prob}\left\{\sum_{t=0}^{N-1} \psi_t \geq \Lambda \sqrt{\sum_{t=0}^{N-1} \sigma_t^2}\right\} \leq \exp\{-\Lambda^2/3\}.$$

To start with, using Lemma 1 with the parameter mapping (13), we have

$$\begin{aligned}
&\frac{1}{1-\beta_s}\big(F(x_{k+1}) - F(x^\star)\big) + \frac{1}{\alpha_s}V_d(x^\star, z_{k+1}) \\
\leq\ &\frac{\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{\alpha_s}V_d(x^\star, z_k) \\
&+ \frac{(\|\nabla f(x_k) - \nabla f_{i_k}(x_k)\|_* + M)^2}{2(\alpha_s^{-1} - L(1-\beta_s))} + \langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\rangle \\
\leq\ &\frac{\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{\alpha_s}V_d(x^\star, z_k) + \frac{M^2}{\alpha_s^{-1} - L(1-\beta_s)} \\
&+ \frac{\|\nabla f(x_k) - \nabla f_{i_k}(x_k)\|_*^2}{\alpha_s^{-1} - L(1-\beta_s)} + \langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\rangle\,.
\end{aligned}$$

Summing the above inequality from $k = sm, \ldots, sm+m-1$ and using the choice $\alpha_s = \frac{\lambda_1}{L(1-\beta_s)}$ with $\lambda_1 \leq \frac{2}{3}$, we obtain

$$\begin{aligned}
&\frac{\alpha_s}{1-\beta_s}\big(F(\tilde{x}_{s+1}) - F(x^\star)\big) + \frac{1}{m}V_d(x^\star, z_{(s+1)m}) \\
\leq\ &\frac{\alpha_s\beta_s}{1-\beta_s}\big(F(\tilde{x}_s) - F(x^\star)\big) + \frac{1}{m}V_d(x^\star, z_{sm}) + 3\alpha_s^2 M^2 \\
&+ \frac{3\alpha_s^2}{m}\sum_{k=sm}^{sm+m-1}\|\nabla f(x_k) - \nabla f_{i_k}(x_k)\|_*^2 + \frac{\alpha_s}{m}\sum_{k=sm}^{sm+m-1}\langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\rangle.
\end{aligned}$$

With our parameter choices, the relations in (15) hold and thus we can telescope the above inequality from $s = S-1, \ldots, 0$,

$$\begin{aligned}
\frac{\alpha_{S-1}}{1-\beta_{S-1}}\big(F(\tilde{x}_S) - F(x^\star)\big) \leq\ &\frac{1}{m}V_d(x^\star, x_0) + 3M^2\sum_{s=0}^{S-1}\alpha_s^2 \\
&+ \frac{3}{m}\underbrace{\sum_{k=0}^{K-1}\alpha_{\lfloor k/m\rfloor}^2\|\nabla f(x_k) - \nabla f_{i_k}(x_k)\|_*^2}_{R_4} + \frac{1}{m}\underbrace{\sum_{k=0}^{K-1}\alpha_{\lfloor k/m\rfloor}\langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\rangle}_{R_5}.
\end{aligned} \tag{16}$$

Denoting $\mathcal{V}_k^2 \triangleq \|\nabla f(x_k) - \nabla f_{i_k}(x_k)\|_*^2$, $\bar{\alpha} = \sum_{k=0}^{K-1} \alpha^2_{\lfloor k/m \rfloor} = m \sum_{s=0}^{S-1} \alpha_s^2$, for R_4, by Jensen's inequality, we have

$$\mathbb{E}\left[\exp\left\{\frac{1}{\bar{\alpha}}\sum_{k=0}^{K-1}\alpha^2_{\lfloor k/m \rfloor}\mathcal{V}_k^2/\sigma^2\right\}\right] \leq \frac{1}{\bar{\alpha}}\sum_{k=0}^{K-1}\alpha^2_{\lfloor k/m \rfloor}\mathbb{E}\left[\exp\left\{\mathcal{V}_k^2/\sigma^2\right\}\right] \overset{(\star)}{\leq} \exp\{1\},$$

where $(\star)$ uses the additional assumption $\mathbb{E}_{i_k}\left[\exp\left\{\mathcal{V}_k^2/\sigma^2\right\}\right] \leq \exp\{1\}$.

Then, based on Markov's inequality, we have for any $\Lambda \geq 0$,

$$\begin{aligned}\text{Prob}\left\{\exp\left\{\frac{1}{\bar{\alpha}}\sum_{k=0}^{K-1}\alpha^2_{\lfloor k/m \rfloor}\mathcal{V}_k^2/\sigma^2\right\} \geq \exp\{\Lambda+1\}\right\} &\leq \exp\{-\Lambda\},\\ \text{Prob}\left\{R_4 \geq (\Lambda+1)\sigma^2 m\sum_{s=0}^{S-1}\alpha_s^2\right\} &\leq \exp\{-\Lambda\}. \end{aligned} \tag{17}$$

For R_5, since we have $\mathbb{E}_{i_k}\left[\alpha_{\lfloor k/m \rfloor}\left\langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\right\rangle\right] = 0$ and

$$\mathbb{E}_{i_k}\left[\exp\left\{\frac{\alpha^2_{\lfloor k/m \rfloor}\left\langle \nabla f(x_k) - \nabla f_{i_k}(x_k), z_k - x^\star\right\rangle^2}{\alpha^2_{\lfloor k/m \rfloor}\sigma^2 D_X^2}\right\}\right] \leq \mathbb{E}_{i_k}\left[\exp\left\{\mathcal{V}_k^2/\sigma^2\right\}\right] \leq \exp\{1\},$$

which is based on the "light tail" assumption, using Lemma 2, we obtain

$$\text{Prob}\left\{R_5 \geq \Lambda\sigma D_X\sqrt{m\sum_{s=0}^{S-1}\alpha_s^2}\right\} \leq \exp\{-\Lambda^2/3\}. \tag{18}$$

Combining (16), (17) and (18), based on the parameter setting and using the notation

$$\begin{aligned}\mathcal{K}_0(m) &\triangleq \frac{6LmV_d(x^\star,x_0)}{(K+m)^2} + \frac{8\sqrt{2V_d(x^\star,x_0)}\sqrt{\sigma^2+M^2}}{\sqrt{K+m}},\\ R_6 &\triangleq \frac{12L\sigma^2}{\lambda_1(S+1)^2}\sum_{s=0}^{S-1}\alpha_s^2 + \frac{4L\sigma D_X}{\lambda_1(S+1)^2\sqrt{m}}\sqrt{\sum_{s=0}^{S-1}\alpha_s^2},\end{aligned}$$

we conclude that

$$\text{Prob}\left\{F(\tilde{x}_S) - F(x^\star) \leq \mathcal{K}_0(m) + \Lambda R_6\right\} \geq 1 - (\exp\{-\Lambda^2/3\} + \exp\{-\Lambda\}).$$

For R_6, using the choice of α_s and λ_1, we obtain

$$R_6 \leq \frac{4\sqrt{6}\sigma D_X}{3\sqrt{K+m}} + \frac{8\lambda_1\sigma^2(S+1)}{L} \leq \frac{4\sqrt{6}\sigma D_X}{3\sqrt{K+m}} + \frac{4\sqrt{2V_d(x^\star,x_0)}\sigma^2}{\sqrt{K+m}\sqrt{\sigma^2+M^2}} \leq \frac{4\sqrt{6}\sigma\left(\sqrt{3V_d(x^\star,x_0)} + D_X\right)}{3\sqrt{K+m}},$$

which completes the proof.

B.2.4 Proof of Theorem 2

Using Assumption (c), Lemma 1 with

$$\begin{cases} x = x_k^{j_k} \\ z = z_k \\ z^+ = z_{k+1} \\ y = \phi_{j_k}^k \\ y^+ = \phi_{j_k}^{k+1} \\ \alpha = \alpha_k \\ \beta = \beta_k \end{cases},$$

and taking expectation, if $\alpha_k(1-\beta_k) < 1/L$, we have

$$\begin{aligned} &\frac{1}{1-\beta_k}\mathbb{E}_{i_k,j_k}\left[F(\phi_{j_k}^{k+1}) - F(x^\star)\right] + \frac{1}{\alpha_k}\mathbb{E}_{i_k,j_k}\left[V_d(x^\star, z_{k+1})\right] \\ &\leq \frac{\beta_k}{1-\beta_k}\mathbb{E}_{j_k}\left[F(\phi_{j_k}^k) - F(x^\star)\right] + \frac{1}{\alpha_k}V_d(x^\star, z_k) + \frac{(\sigma+M)^2}{2(\alpha_k^{-1} - L(1-\beta_k))}. \end{aligned} \tag{19}$$

Note that

$$\begin{aligned} \mathbb{E}_{i_k,j_k}\left[F(\phi_{j_k}^{k+1}) - F(x^\star)\right] &= \mathbb{E}_{i_k,j_k}\left[\sum_{j=1}^m \left(F(\phi_j^{k+1}) - F(x^\star)\right) - \sum_{j\neq j_k}^m \left(F(\phi_j^k) - F(x^\star)\right)\right] \\ &= \mathbb{E}_{i_k,j_k}\left[\sum_{j=1}^m \left(F(\phi_j^{k+1}) - F(x^\star)\right)\right] - \mathbb{E}_{j_k}\left[\sum_{j\neq j_k}^m \left(F(\phi_j^k) - F(x^\star)\right)\right]. \end{aligned}$$

Dividing both sides of (19) by m and then adding $\frac{1}{(1-\beta_k)m}\mathbb{E}_{j_k}\left[\sum_{j\neq j_k}^m \left(F(\phi_j^k) - F(x^\star)\right)\right]$ to both sides, we obtain

$$\begin{aligned} &\frac{1}{1-\beta_k}\mathbb{E}_{i_k,j_k}\left[\frac{1}{m}\sum_{j=1}^m F(\phi_j^{k+1}) - F(x^\star)\right] + \frac{1}{\alpha_k m}\mathbb{E}_{i_k,j_k}\left[V_d(x^\star, z_{k+1})\right] \\ \leq\ & -\frac{1}{m}\mathbb{E}_{j_k}\left[F(\phi_{j_k}^k) - F(x^\star)\right] + \frac{1}{1-\beta_k}\left(\frac{1}{m}\sum_{j=1}^m F(\phi_j^k) - F(x^\star)\right) + \frac{1}{\alpha_k m}V_d(x^\star, z_k) \\ &+ \frac{(\sigma+M)^2}{2m(\alpha_k^{-1} - L(1-\beta_k))} \\ =\ & \frac{1 - \frac{1-\beta_k}{m}}{1-\beta_k}\left(\frac{1}{m}\sum_{j=1}^m F(\phi_j^k) - F(x^\star)\right) + \frac{1}{\alpha_k m}V_d(x^\star, z_k) + \frac{(\sigma+M)^2}{2m(\alpha_k^{-1} - L(1-\beta_k))}. \end{aligned} \tag{20}$$

It can be verified that with our parameters choice: $\beta_k = \frac{k/m}{k/m+2}$ and $\alpha_k = \frac{\lambda_2}{L(1-\beta_k)}$, the following holds for $k \geq 0$,

$$\alpha_{k+1}\frac{1 - \frac{1-\beta_{k+1}}{m}}{1-\beta_{k+1}} \leq \frac{\alpha_k}{1-\beta_k} \text{ and } \beta_0 = 0.$$

Then, we can telescope (20) from $k = K-1, \ldots, 0$, which results in

$$\begin{aligned} &\frac{\alpha_{K-1}}{1-\beta_{K-1}}\mathbb{E}\left[\frac{1}{m}\sum_{j=1}^m F(\phi_j^K) - F(x^\star)\right] + \frac{1}{m}\mathbb{E}\left[V_d(x^\star, z_K)\right] \\ &\leq \frac{\lambda_2(m-1)}{Lm}\big(F(x_0) - F(x^\star)\big) + \frac{1}{m}V_d(x^\star, x_0) + \sum_{k=0}^{K-1}\frac{\alpha_k(\sigma+M)^2}{2m(\alpha_k^{-1} - L(1-\beta_k))}. \end{aligned}$$

Using the definition of $\bar{\phi}^K$ and convexity, we obtain

$$\begin{aligned}
\mathbb{E}\left[F(\bar{\phi}^K) - F(x^\star)\right] &\leq \frac{1-\beta_{K-1}}{\alpha_{K-1}}\left(\frac{\lambda_2(m-1)}{Lm}\big(F(x_0)-F(x^\star)\big) + \frac{1}{m}V_d(x^\star,x_0)\right) \\
&\quad + \frac{1-\beta_{K-1}}{\alpha_{K-1}}\sum_{k=0}^{K-1}\frac{\alpha_k(\sigma+M)^2}{2m(\alpha_k^{-1}-L(1-\beta_k))} \\
&\overset{(a)}{=} \frac{4(m-1)\big(F(x_0)-F(x^\star)\big)}{m\left(\frac{K-1}{m}+2\right)^2} + \frac{4LV_d(x^\star,x_0)}{\lambda_2 m\left(\frac{K-1}{m}+2\right)^2} \\
&\quad + \frac{3\lambda_2(\sigma+M)^2}{2Lm\left(\frac{K-1}{m}+2\right)^2}\sum_{k=0}^{K-1}\left(\frac{k}{m}+2\right)^2 \\
&\overset{(b)}{\leq} \frac{4(m-1)\big(F(x_0)-F(x^\star)\big)}{m\left(\frac{K-1}{m}+2\right)^2} + \frac{4LV_d(x^\star,x_0)}{\lambda_2 m\left(\frac{K-1}{m}+2\right)^2} + \frac{4\lambda_2(\sigma+M)^2\left(\frac{K-1}{m}+2\right)}{L}, \qquad (21)
\end{aligned}$$

where (a) uses $\lambda_2 \leq \frac{2}{3}$, (b) follows from simple integration arguments and that $\frac{K}{m}+2 \leq 2\left(\frac{K-1}{m}+2\right)$ since $K \geq 1, m \geq 1$.

Based on the choice of

$$\lambda_2 = \min\left\{\frac{2}{3}, \frac{L\sqrt{V_d(x^\star,x_0)}}{\sqrt{m}(\sigma+M)\left(\frac{K-1}{m}+2\right)^{\frac{3}{2}}}\right\},$$

(21) can be further upper bounded as

$$\mathbb{E}\left[F(\bar{\phi}^K) - F(x^\star)\right] \leq \frac{4(m-1)\big(F(x_0)-F(x^\star)\big)}{m\left(\frac{K-1}{m}+2\right)^2} + \frac{6LV_d(x^\star,x_0)}{m\left(\frac{K-1}{m}+2\right)^2} + \frac{8\sqrt{V_d(x^\star,x_0)}(\sigma+M)}{m^{\frac{1}{2}}\left(\frac{K-1}{m}+2\right)^{\frac{1}{2}}}.$$

B.3 Connections between AM1-SGD and Katyusha

The high level idea of Katyusha momentum is that it works as a "magnet" inside an epoch of SVRG updates, which "stabilizes" the iterates so as to make Nesterov's momentum effective (Allen-Zhu, 2018). In theory, the key effect of Katyusha momentum is that it allows the tightest possible variance bound for the stochastic gradient estimator of SVRG (cf. Lemma 2.4 and its comments in Allen-Zhu (2018)). In this sense, we can interpret Katyusha momentum as a variance reducer that further reduces the variance of SVRG. Below we show the similarity between the construction of Katyusha and AM1-SGD, based on which we conjecture that the amortized momentum can also reduce the variance of SGD (and thus increase the robustness). However, in theory, following a similar analysis of Katyusha, we cannot guarantee a reduction of the variance in the worst case.

Deriving AM1-SGD from Katyusha For simplicity, we consider the Euclidean setting (Algorithm 2). Katyusha has the following scheme (non-proximal, in the original notations, σ is the strong convexity parameter, cf. Algorithm 1 with Option I in Allen-Zhu (2018))[8]:

Initialize: $\widetilde{x}^0 = y_0 = z_0 = x_0, \eta = \frac{1}{3L}, \omega = 1 + \alpha\sigma$.
for $s = 0, \dots, S-1$ **do**
  Compute and store $\nabla f(\widetilde{x}^s)$.
  for $j = 0, \dots, m-1$ **do**
    $k = sm + j$.
    $x_k = \tau_1 \cdot z_k + \tau_2 \cdot \widetilde{x}^s + (1 - \tau_1 - \tau_2) \cdot y_k$.
    $\widetilde{\nabla}_k = \nabla f_{i_k}(x_k) - \nabla f_{i_k}(\widetilde{x}^s) + \nabla f(\widetilde{x}^s)$.
    $z_{k+1} = z_k - \alpha \cdot \widetilde{\nabla}_k$.
    $y_{k+1} = x_k - \eta \cdot \widetilde{\nabla}_k$.
  end for
  $\widetilde{x}^{s+1} = \left(\sum_{j=0}^{m-1} \omega^j\right)^{-1} \cdot \sum_{j=0}^{m-1} \omega^j \cdot y_{sm+j+1}$.
end for
Output: $\widetilde{x}^S$.

We can eliminate the sequence $\{z_k\}$ in this scheme. Note that in the parameter setting of Katyusha, we have $\eta = \alpha\tau_1$, and thus

$$\begin{aligned} x_{k+1} &= \tau_1 \cdot z_{k+1} + \tau_2 \cdot \widetilde{x}^s + (1 - \tau_1 - \tau_2) \cdot y_{k+1} \\ &= \tau_1 \cdot z_k - \eta \cdot \widetilde{\nabla}_k + \tau_2 \cdot \widetilde{x}^s + (1 - \tau_1 - \tau_2) \cdot y_k + (1 - \tau_1 - \tau_2) \cdot (y_{k+1} - y_k) \\ &= x_k - \eta \cdot \widetilde{\nabla}_k + (1 - \tau_1 - \tau_2) \cdot (y_{k+1} - y_k) \\ &= y_{k+1} + (1 - \tau_1 - \tau_2) \cdot (y_{k+1} - y_k). \end{aligned}$$

Hence, the inner loops can be written as

$$\begin{aligned} y_{k+1} &= x_k - \eta \cdot \widetilde{\nabla}_k, \\ x_{k+1} &= y_{k+1} + (1 - \tau_1 - \tau_2) \cdot (y_{k+1} - y_k), \end{aligned}$$

which is the Nesterov's scheme (scheme (4)). At the end of each inner loop (when $k = sm + m - 1$),

$$x_{(s+1)m} = \tau_1 \cdot z_{(s+1)m} + \tau_2 \cdot \widetilde{x}^s + (1 - \tau_1 - \tau_2) \cdot y_{(s+1)m},$$

while at the beginning of the next inner loop,

$$x_{(s+1)m} = \tau_1 \cdot z_{(s+1)m} + \tau_2 \cdot \widetilde{x}^{s+1} + (1 - \tau_1 - \tau_2) \cdot y_{(s+1)m},$$

which means that we need to set $x_{(s+1)m} \leftarrow x_{(s+1)m} + \tau_2 \cdot (\widetilde{x}^{s+1} - \widetilde{x}^s)$ (reassign the value of $x_{(s+1)m}$).

Then, the following is an equivalent scheme of Katyusha:

[8] We change the notation x_{k+1} to x_k.

Initialize: $\widetilde{x}^0 = y_0 = x_0, \eta = \frac{1}{3L}, \omega = 1 + \alpha\sigma$.

for $s = 0, \ldots, S-1$ **do**
  for $j = 0, \ldots, m-1$ **do**
    $k = sm + j$.
    $y_{k+1} = x_k - \eta \cdot \widetilde{\nabla}_k$.
    $x_{k+1} = y_{k+1} + (1 - \tau_1 - \tau_2) \cdot (y_{k+1} - y_k)$.
  end for
  $\widetilde{x}^{s+1} = \left(\sum_{j=0}^{m-1} \omega^j\right)^{-1} \cdot \sum_{j=0}^{m-1} \omega^j \cdot y_{sm+j+1}$.
  $x_{(s+1)m} \leftarrow x_{(s+1)m} + \tau_2 \cdot (\widetilde{x}^{s+1} - \widetilde{x}^s)$.
end for

Output: $\widetilde{x}_S$.

Now it is clear that the inner loops use Nesterov's momentum and the Katyusha momentum is injected for every m iterations. If we replace the SVRG estimator $\widetilde{\nabla}_k$ with $\nabla f_{i_k}(x_k)$, set $1 - \tau_1 - \tau_2 = 0$, which is to eliminate Nesterov's momentum, and use a uniform average for $\widetilde{x}^{s+1}$, the above scheme becomes exactly AM1-SGD (Algorithm 2).

If we only replace the SVRG estimator $\widetilde{\nabla}_k$, the scheme can be regarded as adding amortized momentum to M-SGD. This scheme requires tuning the ratio of Nesterov's momentum and amortized momentum. In our preliminary experiments, after suitable tuning, we observed some performance improvement. However, this scheme increases the complexity, which we do not consider it worthwhile.

A recent work (Zhou et al., 2018) shows that when $1 - \tau_1 - \tau_2 = 0$, which is to solely use Katyusha momentum, one can still derive optimal rates and the algorithm is greatly simplified. Their proposed algorithm (i.e., MiG) is structurally more similar to AM1-SGD.

C Training evaluation

Due to the mechanism of back-propagation, evaluating train-batch loss basically incurs no overhead. It can be efficiently used to indicate the training progress. However, it can only be treated as a coarse approximation to the full-batch loss as shown in Figure 2b. If batch normalization (Ioffe and Szegedy, 2015) or dropout (Srivastava et al., 2014) is used in training, the model changes during the training phase, which makes train-batch loss less accurate. More importantly, train-batch loss is always observed to be statistically stable, which omits many important characteristics of an optimizer such as robustness, oscillations, etc. We include a comparison of train-batch loss and full-batch loss on training ResNet18 with pre-activation on CIFAR-10 (the experiment in Appendix A.3) in Figure 17.

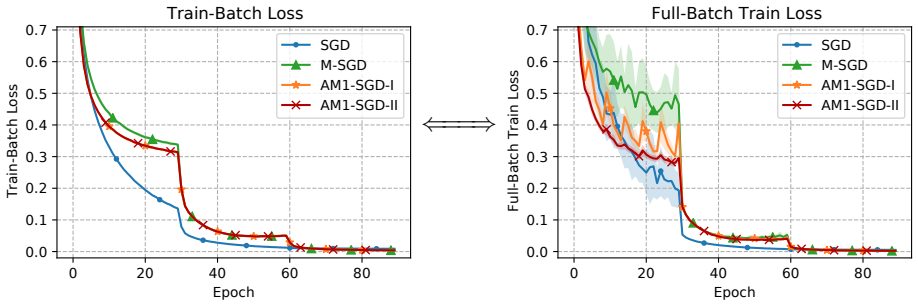


Figure 17: Train-batch loss vs. full-batch loss.

Moreover, train-batch loss does not capture the effect of different output options. We observed that sometimes for different optimizers, even if their convergences on train-batch loss are indistinguishable, their convergences on test accuracy can vary greatly. We show two examples in Figure 18, where the ImageNet experiment is from Section 6 and the CIFAR-10 experiment is from Table 4 with $m = 10$.

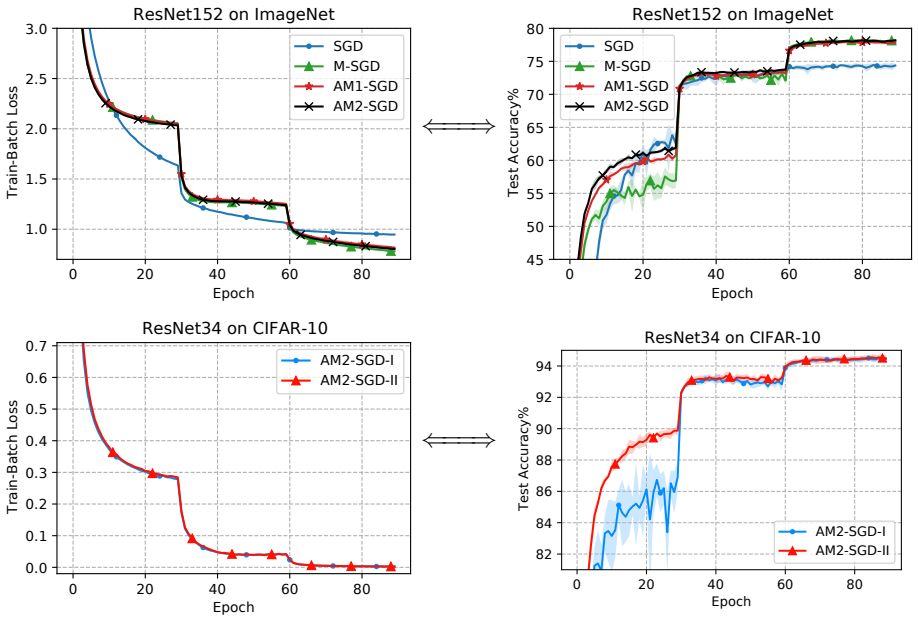


Figure 18: Train-batch loss vs. test accuracy.

D Experimental Setup

All of our experiments were conducted using PyTorch (Paszke et al., 2017) library.

D.1 Classification Setup

CIFAR-10 & CIFAR-100 Our implementation (e.g., ResNet and DenseNet implementations, data pre-processing) generally follows the repository `https://github.com/kuangliu/pytorch-cifar`. All the CIFAR experiments in this paper used a single GPU in a mix of RTX2080Ti, TITAN Xp and TITAN V. The batch size is fixed to 128. We used cross-entropy loss with 0.0005 weight decay and used batch normalization (Ioffe and Szegedy, 2015). Data augmentation includes random 32-pixel crops with a padding of 4-pixel and random horizontal flips with 0.5 probability. We used step (or multi-step) learning rate scheduler with a decay factor 10. For the CIFAR-10 experiments, we trained 90 epochs and decayed the learning rate every 30 epochs. For the CIFAR-100 experiments, we trained 300 epochs and decayed the learning rate at 150 epoch and 225 epoch following the settings in DenseNet (Huang et al., 2017).

ImageNet In the ImageNet experiments, we tried both ResNet50 and ResNet152 (He et al., 2016a). The training strategy is the same as the PyTorch's official repository `https://github.com/pytorch/examples/tree/master/imagenet`, which uses a batch size of 256. The learning rate starts at 0.1 and decays by a factor of 10 every 30 epochs. Also, we applied weight decay with 0.0001 decay rate to the model during the training. For the data augmentation, we applied random 224-pixel crops and random horizontal flips with 0.5 probability. Here, we ran all experiments across 8 NVIDIA P100 GPUs for 90 epochs.

D.2 Language Model Setup

We followed the implementation in the repository `https://github.com/salesforce/awd-lstm-lm` and trained word level Penn Treebank with LSTM without fine-tuning or continuous cache pointer augmentation for 750 epochs. The experiments were conducted on a single RTX2080Ti. We used the default hyper-parameter tuning except for learning rate and momentum: The LSTM has 3 layers containing 1150 hidden units each, embedding size is 400, gradient clipping has a maximum norm 0.25, batch size is 80, using variable sequence length, dropout for the layers has probability 0.4, dropout for the RNN layers has probability 0.3, dropout for the input embedding layer has probability 0.65, dropout to remove words from embedding layer has probability 0.1, weight drop (Merity et al., 2018) has probability 0.5, the amount of $\ell 2$-regularization on the RNN activation is 2.0, the amount of slowness regularization applied on the RNN activation is 1.0 and all weights receive a weight decay of 0.0000012.

References

Allen-Zhu, Z. (2018). Katyusha: The First Direct Acceleration of Stochastic Gradient Methods. *J. Mach. Learn. Res.*, 18(221):1–51.

Auslender, A. and Teboulle, M. (2006). Interior gradient and proximal methods for convex and conic optimization. *SIAM J. Optim.*, 16(3):697–725.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Ghadimi, S. and Lan, G. (2012). Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization i: A generic algorithmic framework. *SIAM J. Optim.*, 22(4):1469–1492.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *CVPR*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, pages 4700–4708.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Lan, G. (2012). An optimal method for stochastic composite optimization. *Math. Program.*, 133(1-2):365–397.

Lan, G., Nemirovski, A., and Shapiro, A. (2012). Validation analysis of mirror descent stochastic approximation method. *Math. Program.*, 134(2):425–458.

Loshchilov, I. and Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*.

Lucas, J., Sun, S., Zemel, R., and Grosse, R. (2019). Aggregated Momentum: Stability Through Passive Damping. In *ICLR*.

Ma, J. and Yarats, D. (2019). Quasi-hyperbolic momentum and Adam for deep learning. In *ICLR*.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and Optimizing LSTM Language Models. In *ICLR*.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.

Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Comput. Math. & Math. Phys.*, 4(5):1–17.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Zhou, K., Shang, F., and Cheng, J. (2018). A Simple Stochastic Variance Reduced Algorithm with Fast Convergence Rates. In *ICML*, pages 5980–5989.