

---

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

---

Kaiwen Zhou<sup>1</sup>

Yanghua Jin<sup>2</sup>

Qinghua Ding<sup>1</sup>

James Cheng<sup>1</sup>

<sup>1</sup> Department of Computer Science and Engineering, The Chinese University of Hong Kong

<sup>2</sup> Preferred Networks, Inc.

kwzhou@cse.cuhk.edu.hk, jinhy@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  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , 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<sup>1</sup> (constant step) (Nesterov, 2013b) ( $y \in \mathbb{R}^n$ ,  $\eta, \beta \in \mathbb{R}$  and  $y_0 = x_0$ ):

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

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

<sup>1</sup>We exchange the notations of  $x$  and  $y$  in Nesterov (2013b).

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<sup>2</sup> 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.

<sup>2</sup>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  $\mathbb{E}_{i_k} [\cdot] \triangleq \mathbb{E} [\cdot | (i_0, \dots, i_{k-1})]$ .

**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  $x^* \in X$  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*  $d : X \rightarrow \mathbb{R}$  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*,  $\text{Prox}_h(x, \mathcal{G}) \triangleq \arg \min_{u \in X} \{V_d(u, x) + \langle \mathcal{G}, u \rangle + h(u)\}$ , 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:**  $\tilde{x}_0 = z_0 = x_0, S = K/m$ .<sup>3</sup>

- 1: **for**  $s = 0, \dots, S - 1$  **do**
- 2:   **for**  $j = 0, \dots, m - 1$  **do**
- 3:      $k = sm + j$ .
- 4:      $x_k = (1 - \beta_s) \cdot z_k + \beta_s \cdot \tilde{x}_s$ .
- 5:      $z_{k+1} = \text{Prox}_{\alpha_s h}(z_k, \alpha_s \cdot \nabla f_{i_k}(x_k))$ .
- 6:   **end for**
- 7:      $\tilde{x}_{s+1} = \frac{1 - \beta_s}{m} \cdot \sum_{j=1}^m z_{sm+j} + \beta_s \cdot \tilde{x}_s$ .
- 8: **end for**

**Output:**  $\tilde{x}_S$ .

---

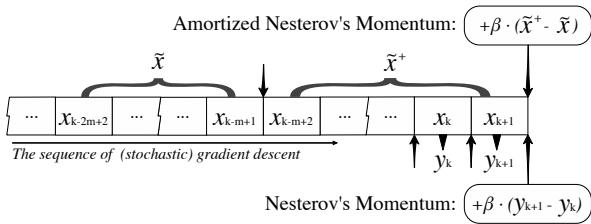


Figure 1: Graphical illustration of Amortized Nesterov’s Momentum. This figure describes how the momentum is injected into the sequence of gradient descent  $\{x_k\}$ .

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  $\eta$  and  $\beta$  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

<sup>3</sup>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  $\eta$ , momentum  $\beta$ , amortization length  $m$ , iteration number  $K$ .  
**Initialize:**  $x \leftarrow x_0, \tilde{x} \leftarrow x_0, \tilde{x}^+ \leftarrow 0$ .

- 1: **for**  $k = 0, \dots, K - 1$  **do**
- 2:    $x \leftarrow x - \eta \cdot \nabla f_{i_k}(x)$ .
- 3:    $\tilde{x}^+ \leftarrow \tilde{x}^+ + \frac{1}{m} \cdot x$ .
- 4:   **if**  $(k + 1) \bmod m = 0$  **then**  
        // adding amortized momentum.
- 5:      $x \leftarrow x + \beta \cdot (\tilde{x}^+ - \tilde{x})$ .
- 6:      $\tilde{x} \leftarrow \tilde{x}^+, \tilde{x}^+ \leftarrow 0$ .
- 7:   **end if**
- 8: **end for**

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

---

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).

**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  $\phi = [\phi_1 \dots \phi_m] \in E^m$ , 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:     $z_{k+1} = \text{Prox}_{\alpha_k h}(z_k, \alpha_k \cdot \nabla f_{i_k}(x_k^{j_k}))$ .
- 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:**  $\bar{\phi}^K = \frac{1}{m} \sum_{j=1}^m \phi_j^K$ .

---

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  $\phi$  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  $(\eta, \beta, m)$ , 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  $\{j_k\}$ .

---

**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  $\eta$ , momentum  $\beta$ , 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:     $x_{k+1} = \phi_{j_{k+1}}^{k+1} + \beta \cdot (\phi_{j_{k+1}}^{k+1} - \phi_{j_k}^k)$ .
- 5:    **if** Option II **then**  $\bar{\phi}^{k+1} = \bar{\phi}^k + \frac{1}{m} \cdot (\phi_{j_k}^{k+1} - \phi_{j_k}^k)$ .
- 6: **end for**

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

---

## 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):

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

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  $\nabla f_i$ , 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)  $0 \leq f(y) - f(x) - \langle \nabla f(x), y - x \rangle \leq \frac{L}{2} \|y - x\|^2 + M \|y - x\|, \forall x, y \in X$ .<sup>4</sup>
- (b)  $\mathbb{E}_i [\nabla f_i(x)] = \nabla f(x), \forall x \in X$ .
- (c)  $\mathbb{E}_i [\|\nabla f_i(x) - \nabla f(x)\|_*^2] \leq \sigma^2, \forall x \in X$ .

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

<sup>4</sup>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:*

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

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  $\lambda_1 = \min \left\{ \frac{2}{3}, \frac{L\sqrt{V_d(x^*, x_0)}}{\sqrt{2m}\sqrt{\sigma^2 + M^2(S+1)^{\frac{3}{2}}}} \right\}$ . Then,*

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

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

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

$$\begin{aligned} \text{Prob}\{F(\tilde{x}_S) - F(x^*) \leq \mathcal{K}_0(m) + \mathcal{K}_1(m, \Lambda)\} \\ \geq 1 - (\exp\{-\Lambda^2/3\} + \exp\{-\Lambda\}), \end{aligned}$$

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

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

**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  $\sigma$  but deteriorates the  $O(L/K^2)$  term (i.e., the acceleration). Note that for AM1-SGD,  $m$  is strictly constrained in  $\{1, \dots, K\}$ . 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  $V_d(x^*, x_0)$ ,  $\sigma$ ,  $L$  and  $M$  could be unknown, especially  $V_d(x^*, x_0)$ . 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  $\lambda_2 = \min \left\{ \frac{2}{3}, \frac{L\sqrt{V_d(x^*, x_0)}}{\sqrt{m}(\sigma+M)(\frac{K-1}{m}+2)^{\frac{3}{2}}} \right\}$ , the output  $\bar{\phi}^K$  satisfies*

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

**Remark:** In comparison with Theorem 1a, Theorem 2 has an additional term  $F(x_0) - F(x^*)$  in the upper bound, which is inevitable. This difference comes from different restrictions on the choice of  $m$ . For AM2-SGD,  $m \geq 1$  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  $m \rightarrow \infty$  to obtain a point table with almost all  $x_0$ , and then the upper bound becomes exactly  $F(x_0) - F(x^*)$ . 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  $\sigma$ , 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

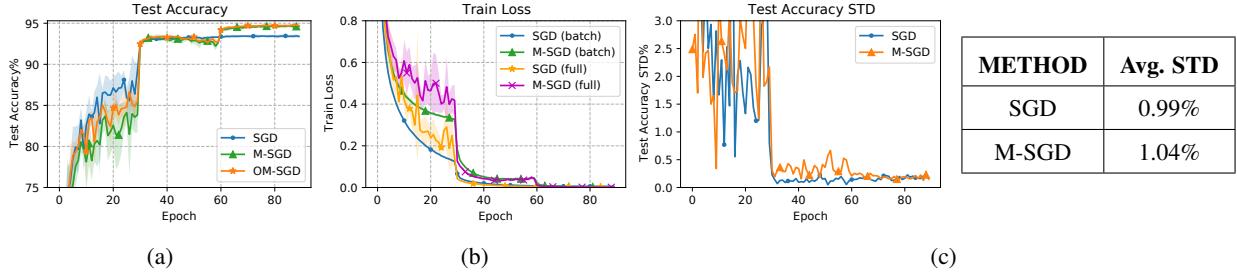


Figure 2: ResNet34 on CIFAR-10. For all methods, initial learning rate  $\eta_0 = 0.1$ , momentum  $\beta = 0.9$ , run 5 seeds (start at same  $x_0$ ). In (a) (b), we plot mean curves with shaded bands indicate  $\pm 1$  standard deviation. (c) shows the standard deviation of test accuracy and its average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

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

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  $k \geq 0$ ,

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

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

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<sup>5</sup> 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<sup>6</sup> (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-

<sup>5</sup>Tensorflow tracks the values  $\{y_k + \beta \cdot v_k\} = \{x_k\}$ .

<sup>6</sup>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.

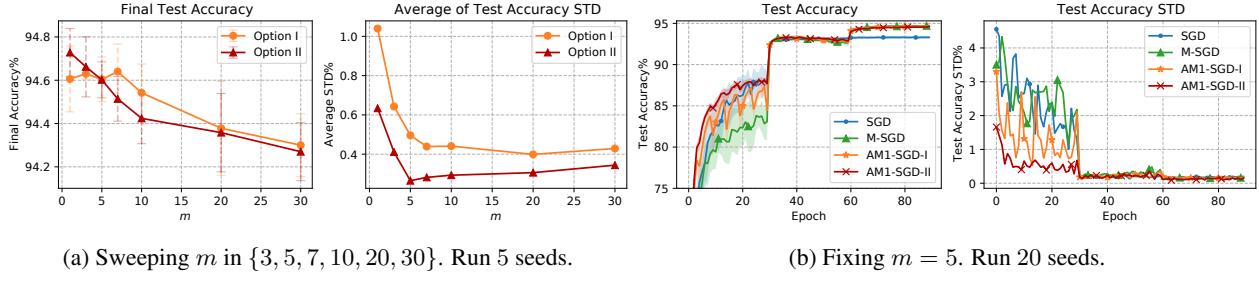


Figure 3: ResNet34 on CIFAR-10. For all methods,  $\eta_0 = 0.1$ ,  $\beta = 0.9$ , using same  $x_0$ . Labels of AM1-SGD are ‘AM1-SGD-{Option}’. Shaded bands (or bars) indicate  $\pm 1$  standard deviation.

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  $\Lambda$ .

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	$94.68\% \pm 0.18\%$	<b>0.59%</b>
AM1-SGD-II	$94.62\% \pm 0.15\%$	<b>0.31%</b>

## 6 EXPERIMENTS

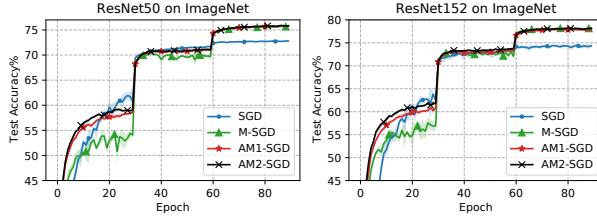
From Theorems 1 and 2, we see that if  $m$  is small, the parameters  $\alpha$  and  $\beta$  do not change a lot from the case where  $m = 1$ . This inspired us to align  $(\eta, \beta)$  of AM1/2-SGD with that of M-SGD and we tune only  $m$ . Such a choice facilitates the usage of AM1/2-SGD. In other words, we used the following parameter settings:  $\eta$  for SGD,  $(\eta, \beta)$  for M-SGD and  $(\eta, \beta, m)$  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 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	<b>75.78%</b> $\pm$ 0.11%	77.82% $\pm$ 0.29%
AM2-SGD	<b>75.85%</b> $\pm$ 0.07%	<b>78.19%</b> $\pm$ 0.15%

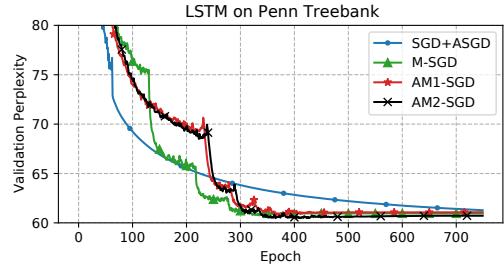
Figure 4 & Table 2: ResNet on ImageNet. Run 3 seeds. Shaded bands indicate  $\pm 1$  standard deviation.

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  $\beta$  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.



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

Figure 5 & Table 3: LSTM on Penn Treebank.

## 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<sup>7</sup> (Polyak and Juditsky, 1992) with constant learning rate 30 as used in Merity et al. (2018). For the choice of  $(\eta, \beta)$ , 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  $\eta$  from  $\{5, 2.5, 1, 0.1, 0.01\}$  and found that  $\eta = 2.5$  resulted in

<sup>7</sup>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  $(\eta, \beta)$  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  $\beta$  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.