Mathematical Programming Computation
https://doi.org/10.1007/s12532-023-00237-5

FULL LENGTH PAPER

Parallel and distributed asynchronous adaptive stochastic gradient methods

Yangyang Xu[1] · Yibo Xu[2] · Yonggui Yan[1] · Colin Sutcher-Shepard[1] · Leopold Grinberg[3] · Jie Chen[4]

Received: 21 January 2021 / Accepted: 27 February 2023

Abstract

Stochastic gradient methods (SGMs) are the predominant approaches to train deep learning models. The adaptive versions (e.g., Adam and AMSGrad) have been extensively used in practice, partly because they achieve faster convergence than the non-adaptive versions while incurring little overhead. On the other hand, asynchronous (async) parallel computing has exhibited significantly higher speed-up over its synchronous (sync) counterpart. Async-parallel non-adaptive SGMs have been well studied in the literature from the perspectives of both theory and practical performance. Adaptive SGMs can also be implemented without much difficulty in an async-parallel way. However, to the best of our knowledge, no theoretical result of async-parallel adaptive SGMs has been established. The difficulty for analyzing adaptive SGMs with async updates originates from the second moment term. In this paper, we propose an async-parallel adaptive SGM based on AMSGrad. We show that the proposed method inherits the convergence guarantee of AMSGrad for both convex and non-convex problems, if the staleness (also called delay) caused by asynchrony is bounded. Our convergence rate results indicate a nearly linear parallelization speed-up if $\tau = o(K^{\frac{1}{4}})$, where τ is the staleness and K is the number of iterations. The proposed method is tested on both convex and non-convex machine learning problems, and the numerical results demonstrate its clear advantages over the sync counterpart and the async-parallel nonadaptive SGM. Our code has been released at https://github.com/RPI-OPT/APAM.

Part of Yibo's work was done when he was a postdoctoral fellow at Rensselaer Polytechnic Institute.

✉ Yangyang Xu
xuy21@rpi.edu

1 Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, USA

2 School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, USA

3 AMD, Cambridge, MA, USA

4 MIT-IBM Watson AI Lab, IBM Research, Armonk, NY, USA

Published online: 19 March 2023

Keywords Stochastic gradient method · Adaptive learning rate · Deep learning

Mathematics Subject Classification 90C15 · 65Y05 · 68W15 · 65K05

1 Introduction

In recent years, *adaptive* stochastic gradient methods (SGMs), such as AdaGrad [12], Adam [19], and AMSGrad [36], have become very popular due to their great success in training deep learning models. These adaptive SGMs can practically be significantly faster than a classic *non-adaptive* SGM. We aim at speeding up adaptive SGMs on massively parallel computing resources. One way is to parallelize them in a *synchronous* (sync) way by using a large batch size, in order to obtain high parallelization speed-up. However, it has been observed [18, 29] that large-batch training in deep learning can often lead to worse generalization than small-batch training. To simultaneously gain fast convergence, high parallelization speed-up, and also good generalization, we propose to develop *asynchronous* (async) parallel *adaptive* SGMs.

Async-parallel computing under either shared-memory or distributed setting has been demonstrated to enjoy significantly higher speed-up than its sync counterpart, e.g., [23, 25, 32, 35]. At each iteration of a sync-parallel method, the workers that finish tasks earlier must wait for those that finish later. This can result in a lot of idle waiting time. In addition, under a shared-memory setting, all workers access the memory simultaneously, which can cause memory congestion [4], and under a distributed setting, enforcing synchronization is often inefficient due to communication latency. For these reasons, a sync-parallel method may have a very low parallelization speed-up. On the contrary, an async-parallel method does not require all workers to keep the same pace and can eliminate the waiting time and the memory congestion issue. However, it may be difficult to guarantee the convergence of an async-parallel method, because outdated information could be used in updating the variables.

Async-parallel methods have been developed for non-adaptive SGMs, e.g., in [1, 23, 35]. However, a *parallel nonadaptive* SGM may be slower than a *non-parallel adaptive* SGM to reach the same accuracy. Hence, it is important to design a method that can achieve the high speed-up of async-parallel implementation and also the fast convergence of an adaptive SGM. How to guarantee a successful integration remains an open question, although numerical experiments have been conducted to demonstrate the performance of async-parallel adaptive SGMs, e.g., in [11, 16]. The non-triviality lies in the integrated analysis of the second-moment term used in adaptive SGMs. In this work, we give an affirmative answer to the question, by designing an async-parallel adaptive SGM under both shared-memory and distributed settings.

1.1 Proposed algorithm

We consider the stochastic program

$$F^* = \underset{\mathbf{x}\in X}{\text{minimize}} \; F(\mathbf{x}) := \mathbb{E}_\xi\big[f(\mathbf{x};\xi)\big], \tag{1.1}$$

where $\xi \in \Xi$ is a random variable, and $X \subseteq \mathbb{R}^n$ is a closed convex set. When ξ is uniformly distributed on a finite set $\Xi = \{\xi_1, \ldots, \xi_N\}$, (1.1) reduces to a finite-sum structured problem, which includes as examples all machine learning problems with pre-collected training data.

For solving (1.1), we propose an async-parallel adaptive SGM, named APAM, which is based on AMSGrad in [36]. We adopt a master-worker set-up. The pseudocode is shown in Algorithm 1, which is from the master's view. The updates in (1.2) through (1.5) are performed by the master, while the workers compute the stochastic gradients $\{\mathbf{g}^{(k)}\}$. Due to the potential information delay caused by asynchrony, $\mathbf{g}^{(k)}$ may not be evaluated at $\mathbf{x}^{(k)}$; see more discussions in Sect. 2.2. In (1.5), we define a weighted norm as $\|\mathbf{x}\|_{\mathbf{v}}^2 := \mathbf{x}^\top \mathrm{Diag}(\mathbf{v})\mathbf{x}$, and if $X = \mathbb{R}^n$, the update reduces to $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}$, where $\oslash$ denotes component-wise division. The weight vector $\widehat{\mathbf{v}}^{(k)}$ depends on all previous stochastic gradients, and thus the effective learning rate $\alpha_k \mathbf{1} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}$ *adaptively* depends on the gradients.

Algorithm 1: **a**sync-**p**arallel **a**daptive stochastic gradient **m**ethod (APAM) from master's view

Initialization: choose $\mathbf{x}^{(1)} \in X$ and $\beta_1, \beta_2 \in [0, 1)$; set $\mathbf{m}^{(0)} = \mathbf{0}$ and $\mathbf{v}^{(0)} = \widehat{\mathbf{v}}^{(0)} = \mathbf{0}$, .

for $k = 1, 2, \ldots$ **do**

Obtain a (possibly outdated) stochastic gradient $\mathbf{g}^{(k)}$ from a worker, and update

$$\mathbf{m}^{(k)} = \beta_1 \mathbf{m}^{(k-1)} + (1 - \beta_1)\mathbf{g}^{(k)}, \tag{1.2}$$

$$\mathbf{v}^{(k)} = \beta_2 \mathbf{v}^{(k-1)} + (1 - \beta_2)\big(\mathbf{g}^{(k)}\big)^2, \tag{1.3}$$

$$\widehat{\mathbf{v}}^{(k)} = \max\big\{\widehat{\mathbf{v}}^{(k-1)}, \mathbf{v}^{(k)}\big\}, \tag{1.4}$$

$$\mathbf{x}^{(k+1)} \in \operatorname*{Arg\,min}_{\mathbf{x} \in X} \langle \mathbf{m}^{(k)}, \mathbf{x} \rangle + \frac{1}{2\alpha_k} \|\mathbf{x} - \mathbf{x}^{(k)}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}}. \tag{1.5}$$

We emphasize the importance of the proposed method in training very large-scale deep learning models. It is well-known that adaptive SGMs converge significantly faster than a non-adaptive SGM; see [12, 19] for example or our numerical results in Sect. 5. In addition, an async-parallel method can achieve much higher speed-up than its sync counterpart. Hence, it is paramount to design a method that can inherit advantages from both adaptiveness and async-parallelization, in order to efficiently train a very "big" deep learning model on multi-core or distributed-memory machines.

We make the exploration on async-parallel adaptive SGM based on AMSGrad, because of its simplicity and nice numerical performance. Besides AMSGrad, there are several other adaptive SGMs in the literature, such as AdaGrad [12], RMSProp [40], Adam [19], Padam [46], and AdaFom [8]. While AdaGrad can have guaranteed sublinear convergence, its numerical performance can be significantly worse than AMSGrad, because the former simply uses $\mathbf{g}^{(k)}$ instead of the exponential averaging gradient $\mathbf{m}^{(k)}$ and also its effective learning rate can decay very fast. Adam can perform similarly or slightly better than AMSGrad, but its convergence is not guaranteed even for convex problems, due to a possibly too large learning rate. Padam is a generalized

version of AMSGrad, and the performance of AdaFom is somehow between AdaGrad and AMSGrad. We believe that the convergence of AdaGrad, Padam and AdaFom can be inherited by their async versions.

1.2 Related works

In the literature, there are many works on SGMs. We briefly review those on async-parallel SGMs and adaptive SGMs, which are closely related to our work.

Async-parallel non-adaptive SGM The stochastic approximation method can date back to 1950s [37] for solving a root-finding problem. The SGM, as a first-order stochastic approximation method, has been analyzed for both convex and non-convex problems; see [15, 31, 34] for example. In order to achieve high speed-up, async-parallel SGM and/or distributed SGM with delayed gradient have been developed to solve problems that involve huge amount of data, e.g., in [1, 14, 21, 23, 28, 35]. The work [1] assumes a distributed setting with a central node and analyzes the SGM with delayed stochastic gradients. [23] studies the async-parallel SGM for non-convex optimization under both shared-memory and distributed settings. After obtaining a sample gradient, the shared-memory async-parallel method in [23] needs to perform randomized coordinate update to avoid overwriting, because all threads are allowed to update the variables without coordination to each other. Recht et al. [35] also studies shared-memory async-parallel SGM. It does not require randomized coordinate update. However, its analysis relies on strong convexity of the objective and the assumption that the data involved in every sample function is sparse. Leblond et al. [21] further removes the sparsity requirement by providing an improved analysis for async-parallel stochastic incremental methods. In [21], a novel "after read" approach is introduced to order the iterate and address one independence issue between the random sample and the iterate that is read. [3, 39] adapt the stepsize of the async SGM to the staleness of stochastic gradient, and [24, 43] explore the async SGM under a decentralized setting.

Adaptive SGM Adam [19] is probably the most popular adaptive SGM. It was proposed for convex problems. However, the convergence of Adam is not guaranteed. To address the convergence issue, Reddi et al. [36] makes a modification to the second-moment term in Adam and proposes AMSGrad. It performs almost the same updates as those in (1.2) through (1.5), with the only difference that AMSGrad uses non-fixed weights in computing $\mathbf{m}^{(k)}$, i.e., it lets $\mathbf{m}^{(k)} = \beta_{1,k}\mathbf{m}^{(k-1)} + (1-\beta_{1,k})\mathbf{g}^{(k)}$ for all $k \geq 1$. In order to guarantee sublinear convergence, Reddi et al. [36] requires a diminishing sequence $\{\beta_{1,k}\}$, and to have a rate of $O(1/\sqrt{k})$, $\{\beta_{1,k}\}$ needs to decay as fast as $1/k$. However, Reddi et al. [36] sets $\beta_{1,k} = \beta_1 \in (0, 1)$, $\forall\, k \geq 1$ in all its numerical experiments, and it turned out that the algorithm with a constant weight β_1 could perform significantly better than that with decaying weights. By new analysis, we will show, as a byproduct, that an $O(1/\sqrt{k})$ convergence rate can be achieved even with a constant weight. Later, Tran and Phong [41] proposes AdamX, which is similar to AMSGrad but addresses a flaw in the analysis of AMSGrad. AdamX embeds $\beta_{1,k}$ in

updating $\widehat{\mathbf{v}}^{(k)}$. However, it still requires a decaying $\beta_{1,k}$ to guarantee sublinear convergence. To have nice generalization, Chen and Gu [6] proposes Padam that includes AMSGrad as a special case. It uses $-\mathbf{m}^{(k)} \oslash (\widehat{\mathbf{v}}^{(k)})^p$ as the search direction, where $p \in (0, 0.5]$. When $p = \frac{1}{2}$, Padam reduces to AMSGrad. It was demonstrated that $p = \frac{1}{8}$ could yield the best numerical performance. To avoid extremely large or small learning rates, Luo et al. [27] proposes variants of Adam and AMSGrad by keeping the second-moment term in nonincreasing intervals. Asymptotically, they approach to non-adaptive SGMs. For strongly-convex online optimization, Fang and Klabjan [13] presents a variant of AMSGrad, and Wang et al. [42] proposes SAdam, as a variant of Adam. For non-convex problems, Chen et al. [8] gives a general framework of Adam-type SGMs and establishes convergence rate results. Padam is extended in [46] and a later version [7] of the paper [6] to non-convex cases. Nazari et al. [30] presents a variant of AMSGrad by introducing one more moving-average term in the update of $\widehat{\mathbf{v}}$, and the analysis is conducted for both convex and non-convex problems.

Async-parallel adaptive SGM The async-parallel implementation of AdaGrad is explored in [11]. Experimental results on training deep neural networks are shown to demonstrate the performance of the async-parallel AdaGrad. However, no convergence analysis is given in [11], and in addition, AdaGrad often performs significantly worse than AMSGrad. Guan et al. [16] proposes a delay-compensated asynchronous Adam, which exhibits advantages over an asynchronous nonadaptive SGM for solving deep learning problems. However, the theoretical result in [16] does not guarantee convergence to stationarity but simply implies that the expected value of gradient norm can be bounded.

For the readers' convenience, we compare, in Table 1, APAM to several closely related methods based on a few important ingredients about the algorithms and the targeted problem.

1.3 Contributions

Our contributions are three-fold. *First*, we propose an async-parallel adaptive SGM, named APAM, which is an asynchronous version of AMSGrad in [36]. APAM works under both shared-memory and distributed settings. For both settings, we adopt a master-worker architecture. Only the master updates model parameters, while the workers compute stochastic gradients asynchronously. APAM is lock-free. The master can perform updates while the workers are reading/receiving variables, and also since only the master updates variables, there is no need to lock the writing process. To the best of our knowledge, APAM is the first async-parallel adaptive SGM that maintains the fast convergence of an adaptive SGM and also achieves a high parallelization speed-up. *Secondly*, we analyze the convergence rate of APAM for both convex and non-convex problems. For convex problems, we establish a sublinear convergence result in terms of the objective error, and for non-convex problems, we show a sublinear convergence result in terms of the violation of stationarity. The established results indicate that the staleness τ has little impact on the convergence speed if it is dominated by $K^{\frac{1}{4}}$, where K is the maximum number of iterations. Therefore, if

Table 1 A comparison of ingredients among several algorithms for solving problems in the form of (1.1)

Method	F & Constr. X	Adapt.	Weights $(\beta_{1,k},\ \beta_{2,k})$	Async.	Order of convergence rate
Mirror descent [1]	cvx & Yes	No	–	Yes	$(1+\bar{\tau}^2/\sqrt{K})/\sqrt{K}$
AMSGrad [36]	cvx & Yes	Yes	$\beta_{1,k}=\beta_1/k,\ \beta_1<\sqrt{\beta_2}$	No	$1/\sqrt{K}$
AdamX [41]	cvx & Yes	Yes	$\beta_{1,k}=\beta_1/k,\ \beta_1<\sqrt{\beta_2}$	No	$1/\sqrt{K}$
Padam† [7]	ncvx & No	Yes	$\beta_1<\beta_2^{2p},\ p\in[0,1/2]$	No	$1/K^{3/4-s/2}$
AdaDelay [39]	cvx & Yes	No	–	Yes	$(\sqrt{1+\bar{\tau}}+\bar{\tau}^4/\sqrt{K})/\sqrt{K}$
AsySG-con [23]	noncvx & No	No	–	Yes	$(1+\tau/\sqrt{K})/\sqrt{K}$
AMSGrad & AdaFom [8]	noncvx & No	Yes	Constant or decreasing	No	$(\log K)/\sqrt{K}$
APAM (this paper)	cvx & Yes	Yes	Constant	Yes	$(1+\tau^2/\sqrt{K})/\sqrt{K}$
	noncvx & No	Yes	Constant	Yes	$(1+\tau/K^{1/4}+\tau^2/\sqrt{K})/\sqrt{K}$

In the second column, "F & Constr. X" reflects the underlying assumption on F and feasibility constraint X: "cvx" for convexity, "noncvx" for non-convexity, "yes" for closed convex constraint X, and "no" for unconstrained problems. In the third column, "Adapt." reflects whether the algorithm implements adaptivity. In the fourth column, "Weights" reflects the restriction on the momentum parameters in the adaptive algorithms: "constant" indicates a constant parameter choice (i.e., $(\beta_{1,k},\ \beta_{2,k})=(\beta_1,\beta_2),\forall\, k$), and "decreasing" indicates a decreasing parameter choice. In the fifth column, "Async." reflects whether the algorithm has a convergence guarantee for its asynchronous implementation with delayed gradient information. In the last column, convergence rate results for both convex and non-convex models are listed: τ for the upper bound on the delay and K for the total number of iterations; for convex models, the convergence is measured by the expected objective gap, while for non-convex models, it is measured by the expected stationarity violation. Specifically, Mirror descent [1] assumes $\mathbb{E}[\tau_k]\le\bar{\tau}$; AdaDelay [39] has the assumption that the delay has a bounded expectation $\mathbb{E}[\tau_k]=\bar{\tau}<\infty$ and a bounded second moment $\mathbb{E}[\tau_k^2]=\Omega(\bar{\tau}^2)$

†For the convergence rate of Padam [7], the parameter s relates to the sparsity of stochastic gradients; in the worst case, $s=\frac{1}{2}$, and the rate reduces to $O(1/\sqrt{K})$

$\tau = o(K^{\frac{1}{4}})$, a nearly-linear speed-up can be achieved, and this is demonstrated by numerical experiments. *Thirdly*, over the course of analyzing APAM, we also conduct new convergence analysis for AMSGrad. Our convergence rate results do not require a diminishing sequence to weigh the gradients. In practice, constant weights are almost always adopted. Hence, our results bring the theory closer to practice.

1.4 Notation and outline

We use lower-case bold letter $\mathbf{x}, \mathbf{y}, \ldots$ for vectors. The i-th component of a vector $\mathbf{x}$ is denoted as x_i. For any two vectors $\mathbf{x}$ and $\mathbf{y}$ of the same size, $\mathbf{x} \odot \mathbf{y}$ denotes a vector by component-wise multiplication, and $\mathbf{x} \oslash \mathbf{y}$ denotes a vector by the component-wise division, with $\frac{0}{0} = 0$. For any $\mathbf{v} \geq \mathbf{0}$, $\sqrt{\mathbf{v}}$ or $(\mathbf{v})^{\frac{1}{2}}$ denotes a vector by the component-wise square root. We add a superscript $^{(k)}$ to specify the iterate, i.e., $\mathbf{x}^{(k)}$ denotes the k-th iterate. $\mathrm{Diag}(\mathbf{v})$ denotes the diagonal matrix with $\mathbf{v}$ as the diagonal vector. Given $\mathbf{v} \geq \mathbf{0}$, $\|\mathbf{x}\|_{\mathbf{v}}^2 := \mathbf{x}^\top \mathrm{Diag}(\mathbf{v})\mathbf{x}$, and $\mathrm{Proj}_{X,\mathbf{v}}(\mathbf{x}) := \arg\min_{\mathbf{y}\in X} \|\mathbf{y} - \mathbf{x}\|_{\mathbf{v}}^2$. We use $\|\cdot\|$ for the Euclidean norm of a vector and also the spectral norm of a matrix. $[n]$ denotes $\{1, \ldots, n\}$, and for a subset $A \subseteq [n]$, A^c denotes the complement set of A. $\tilde{\nabla} f(\mathbf{x})$ denotes a subgradient of f at $\mathbf{x}$, and it reduces to the gradient $\nabla f(\mathbf{x})$ if f is differentiable. We let $\mathcal{H}_k$ be the σ-algebra generated by $\{\mathbf{x}^{(t)}\}_{t\leq k}$.

Outline The rest of the paper is outlined as follows. In Sect. 2, we give details on how to implement the proposed algorithm. Convergence analysis is given in Sect. 3 for convex problems and in Sect. 4 for non-convex problems, and numerical results are shown in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Implementation of the proposed method

In this section, we give more details on how to implement Algorithm 1 and also how the delay happens as the workers run asynchronously in parallel.

2.1 Organization of master and workers

We first explain how the master and workers communicate under a shared-memory or distributed setting.

Shared-memory setting Suppose that there are multiple processors and all the data and variables (or model parameters) are stored in a global memory. We assign one or a few as the *master*(*s*). The updates to $\mathbf{x}$, $\mathbf{m}$, $\mathbf{v}$ and $\widehat{\mathbf{v}}$ in Algorithm 1 are all performed by the master(s), while the computation of $\mathbf{g}$ is done by other processors (called *worker*s). See the left of Fig. 1 for an illustration. Every worker reads $\mathbf{x}$ and data from the global memory, computes a stochastic gradient $\mathbf{g}$, and saves it in a pre-assigned memory. If there is a $\mathbf{g}$ that has not been used, then the master acquires it. Otherwise, the master computes one stochastic gradient by itself. We allow more than one processor to serve as the master in case one is not fast enough to digest the $\mathbf{g}$ vectors fed by the workers. In

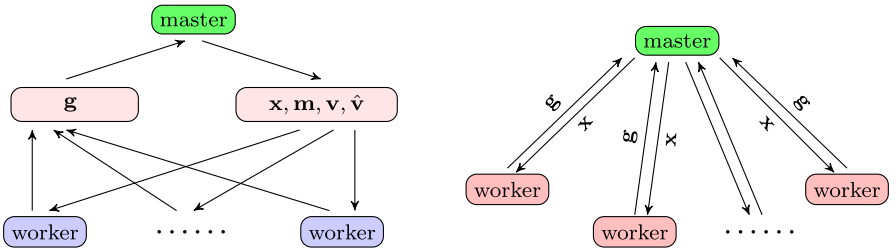


Fig. 1 Shared memory setting (left) vs. distributed setting (right): a demonstration

the case of multiple master processors, we will partition the vectors into blocks and let one master processor update one block, and we synchronize all the master processors while performing the updates. However, we never synchronize the workers.

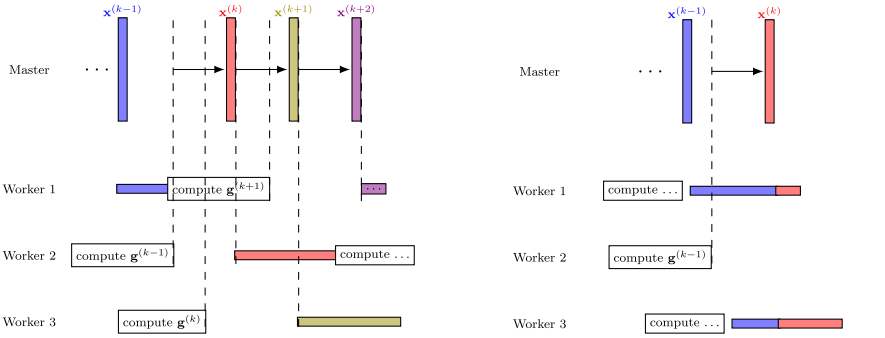
Our shared-memory set-up is fundamentally different from existing ones, e.g., in [21, 23, 35], which allow all processors to update the variables. Without coordination between the processors, overwriting issue will arise if all processors write to the memory at the same time. To avoid the issue, these existing works need to perform randomized coordinate updates [23], or require sparsity of the stochastic gradient [35], or assume strong convexity of the objective [21]. However, in training a deep learning model, neither the sparsity condition nor the strong convexity assumption will hold. In addition, the coordinate update will be inefficient because the whole $\mathbf{g}$ is computed but just one or a few coordinate gradients are used. In contrast, our method does not have this issue due to the master-worker set-up. Furthermore, our set-up enables a simpler analysis without sacrificing the high parallelization speed-up.

Distributed setting Suppose multiple processors do not share memory and hence data need to be transmitted through inter-process communication. The master takes charge of updating $\mathbf{x}$, $\mathbf{m}$, $\mathbf{v}$ and $\widehat{\mathbf{v}}$. It sends $\mathbf{x}$ to workers, and the workers compute and send stochastic gradients to the master for the update. See the right of Fig. 1 for an illustration. We assume that each worker has its own memory and can generate samples by the same distribution.

2.2 Iteration counter and staleness

Notice that Algorithm 1 is viewed from the perspective of the master. We use k as the iteration counter. It increases by one whenever the master performs an update to $\mathbf{x}$. Hence, $\mathbf{x}^{(k)}$ denotes the iterate maintained by the master at the beginning of the k-th update, and $\mathbf{g}^{(k)}$ is the stochastic gradient used in the k-th update. Since the master continuously updates $\mathbf{x}$, after worker $\#i$ reads (or receives) the variable, the master may have already changed $\mathbf{x}$ before it uses the stochastic gradient fed by worker $\#i$. Therefore, the stochastic gradient $\mathbf{g}^{(k)}$ that is used to obtain $\mathbf{x}^{(k+1)}$ may not be evaluated at the current iterate $\mathbf{x}^{(k)}$ but at an outdated one. See Fig. 2 for an illustration. More precisely, we have

$$\mathbf{g}^{(k)} = \frac{1}{b_k} \sum_{i=1}^{b_k} \tilde{\nabla} f(\widehat{\mathbf{x}}^{(k)}; \xi_i^{(k)}), \tag{2.1}$$



(a) **Delayed gradient in a distributed setting.** Each worker uses the $\mathbf{x}$ vector it receives from the master to compute one $\mathbf{g}$ vector. Due to asynchrony, the $\mathbf{g}$ vector that master uses for update may not be calculated at the current $\mathbf{x}$. For example, $\mathbf{g}^{(k+1)}$ is computed by worker 1 at $\mathbf{x}^{(k-1)}$ but used by master to update $\mathbf{x}^{(k+1)}$ to $\mathbf{x}^{(k+2)}$, which causes a delay of 2.

(b) **Inconsistent reading in a shared memory setting.** Each worker reads $\mathbf{x}$ from the shared memory and then computes one $\mathbf{g}$ vector. Due to asynchrony and without lock, the read can be inconsistent. For example, while worker 1 reads $\mathbf{x}^{(k-1)}$, the $\mathbf{x}$ vector is updated by master to $\mathbf{x}^{(k)}$ before worker 1 finishes its reading, and thus worker 1 reads a mixture of $\mathbf{x}^{(k-1)}$ and $\mathbf{x}^{(k)}$.

Fig. 2 A demonstration of consistent but outdated read in the distributed setting (left subfigure) and inconsistent read in the shared memory setting (right subfigure)

where b_k is the number of samples, and $\widehat{\mathbf{x}}^{(k)}$ can be an outdated iterate or a mixture of several iterates; see (2.2) below for its expression.

2.3 Consistent and inconsistent read

In the distributed setting, we have $\widehat{\mathbf{x}}^{(k)} = \mathbf{x}^{k-\tau_k}$ for some $\tau_k \geq 0$ due to communication delay, i.e., the $\mathbf{x}$ received by a worker is a *consistent* but potentially outdated iterate.

#thread	1	2	4	8	16	32
time (sec)	321.0	168.1	88.3	48.3	27.6	19.7

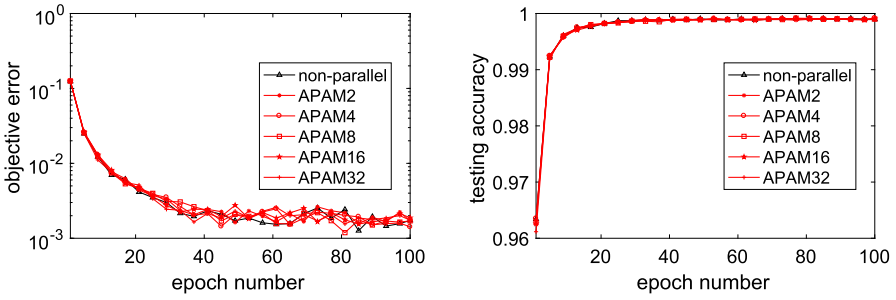


Fig. 3 Running time, objective errors and testing accuracy by APAM with openMP for logistic regression on rcv1 dataset

In the shared-memory setting, since we do not lock $\mathbf{x}$ when a worker computes a stochastic gradient, $\widehat{\mathbf{x}}^{(k)}$ may not be any iterate that ever exists in the memory but is a combination of a few iterates, i.e., the reading is *inconsistent*; see Fig. 2 for an illustration.

Suppose that the read of every coordinate is atomic. Then for each i, it must hold $\widehat{x}_i^{(k)} = x_i^{(k-j)}$ for some integer $j \geq 0$. Let $I_j := \left\{ i : x_i^{(k-j)} = \widehat{x}_i^{(k)} \right\}$ and $\mathcal{I}_j := \cup_{l=0}^{j} I_l$ for each $j \geq 0$. Let $\tau_k = \min \left\{ j : \mathcal{I}_j = [n] \right\}$. Then $\mathcal{I}_{\tau_k} = [n]$, and $\widehat{\mathbf{x}}^{(k)}$ can be formed from $\{\mathbf{x}^{(k-\tau_k)}, \ldots, \mathbf{x}^{(k)}\}$. By the definition of $\mathcal{I}_l$, we have $\mathcal{I}_{l-1} \subseteq \mathcal{I}_l$, and thus

$$\begin{aligned} \widehat{\mathbf{x}}^{(k)} &= \mathbf{x}^{(k)} \odot \mathbf{1}_{\mathcal{I}_0} + \sum_{l=1}^{\tau_k} \mathbf{x}^{(k-l)} \odot (\mathbf{1}_{\mathcal{I}_l} - \mathbf{1}_{\mathcal{I}_{l-1}}) \\ &= \mathbf{x}^{(k)} - \mathbf{x}^{(k)} \odot \mathbf{1}_{\mathcal{I}_0^c} + \sum_{l=1}^{\tau_k} \mathbf{x}^{(k-l)} \odot (\mathbf{1}_{\mathcal{I}_{l-1}^c} - \mathbf{1}_{\mathcal{I}_l^c}) \\ &= \mathbf{x}^{(k)} - \sum_{l=0}^{\tau_k - 1} (\mathbf{x}^{(k-l)} - \mathbf{x}^{(k-l-1)}) \odot \mathbf{1}_{\mathcal{I}_l^c}, \end{aligned} \tag{2.2}$$

where we have used $\mathcal{I}_{\tau_k} = [n]$, and $\mathbf{1}_A$ represents the vector with one at each coordinate $i \in A$ and zero elsewhere. The expression in (2.2) generalizes the relation for atomic lock-free updates in [23, 32]. It follows from (2.2) that

$$\|\widehat{\mathbf{x}}^{(k)} - \mathbf{x}^{(k)}\| \leq \sum_{l=0}^{\tau_k - 1} \left\| (\mathbf{x}^{(k-l)} - \mathbf{x}^{(k-l-1)}) \odot \mathbf{1}_{\mathcal{I}_l^c} \right\| \leq \sum_{l=0}^{\tau_k - 1} \left\| \mathbf{x}^{(k-l)} - \mathbf{x}^{(k-l-1)} \right\|, \tag{2.3}$$

#thread	1	2	4	8	16	32
time (sec)	2396	1259	658	361	201	134

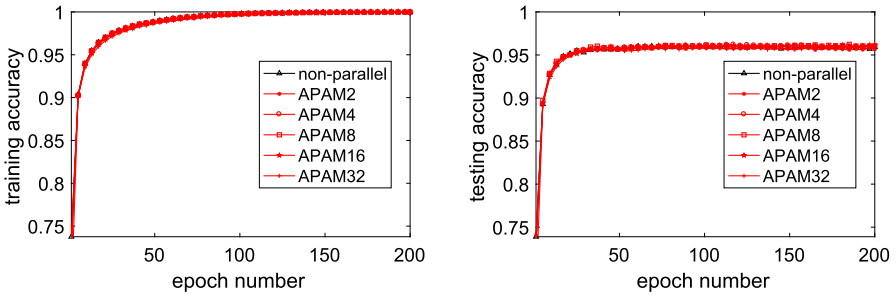


Fig. 4 Running time, training accuracy, and testing accuracy by APAM with openMP for learning a 2-layer fully-connected neural network on MNIST dataset

and

$$\|\widehat{\mathbf{x}}^{(k)} - \mathbf{x}^{(k)}\|^2 \leq \tau_k \sum_{l=0}^{\tau_k - 1} \|\mathbf{x}^{(k-l)} - \mathbf{x}^{(k-l-1)}\|^2. \tag{2.4}$$

These relations are important in our analysis to handle asynchrony.

3 Convergence for convex problems

In this section, we analyze Algorithm 1 for convex problems. Throughout the analysis, we make the following assumptions.

Assumption 1 (Convexity) F in (1.1) is convex, and X is convex and compact.

Under Assumption 1, we define

$$D_\infty = \max_{\mathbf{x},\mathbf{y} \in X} \|\mathbf{x} - \mathbf{y}\|_\infty.$$

Assumption 2 (Bounded gradient in expectation) There is a finite number G_1 such that $\mathbb{E}_\xi \|\tilde{\nabla} f(\mathbf{x}, \xi)\|_1 \leq G_1, \forall \mathbf{x} \in X$.

Assumption 3 (Bounded gradient almost surely) There is a finite number G_∞ such that $\|\tilde{\nabla} f(\mathbf{x}; \xi)\|_\infty \leq G_\infty$, $\forall \mathbf{x} \in X$, and almost surely for all ξ.

Assumption 4 (Unbiased gradient) $\mathbf{g}^{(k)}$ is an unbiased estimate of a subgradient of F at $\widehat{\mathbf{x}}^{(k)}$ for each k, i.e., $\mathbb{E}\big[\mathbf{g}^{(k)} \,|\, \mathcal{H}_k\big] \in \partial F(\widehat{\mathbf{x}}^{(k)})$.

We make a few remarks about the assumptions. The boundedness assumption on X is required to analyze an adaptive SGM for convex problems in existing works, e.g., [12, 19, 36]. Assumption 4 is standard in the analysis of SGMs. It will hold in the distributed setting if data on all workers follow the same distribution and $\{\xi_i^{(k)}\}$ in (2.1) are sampled independently from the distribution. However, in the shared memory setting, the condition can only hold under certain ideal cases when asynchronous updates are performed. Roughly speaking, different realizations of $\{\xi_i^{(k)}\}$ in (2.1) can incur different cost of computing $\mathbf{g}^{(k)}$ and thus affect the iteration counter k, i.e., $\widehat{\mathbf{x}}^{(k)}$ can depend on $\{\xi_i^{(k)}\}$. Hence, the unbiased assumption can hold only if the cost of computing a stochastic subgradient is the same for any realization of ξ and in addition the workers have the same computing power. Leblond et al. [21] addresses the independence issue by an "after read" approach. However, it could be computationally inefficient to first read the entire $\widehat{\mathbf{x}}^{(k)}$ and then sample $\{\xi_i^{(k)}\}$ to compute $\mathbf{g}^{(k)}$. This is also noticed in [21], which implements a different version of its analyzed method. The issue is also addressed in [28], which essentially assumes sparsity of each sample gradient. Both of [21, 28] require strong convexity on the objective. It is unclear whether the issue can be addressed for convex or non-convex cases.

We first establish a couple of lemmas that will be used to show the convergence rate of Algorithm 1 either with delay or without delay. Their proofs are given in the appendix.

Lemma 1 *Let* $\{(\mathbf{x}^{(k)}, \mathbf{m}^{(k)}, \widehat{\mathbf{v}}^{(k)})\}$ *be the sequence from Algorithm 1 with step size sequence* $\{\alpha_k\}$. *Under Assumption 1, it holds for any* $t \geq 1$ *and any* $\mathbf{x} \in X$ *that*

$$\begin{aligned}&(1-\beta_1)\sum_{k=1}^{t}\left(\sum_{j=k}^{t}\alpha_j\beta_1^{j-k}\right)\left\langle \mathbf{x}^{(k)}-\mathbf{x}, \mathbf{g}^{(k)}\right\rangle \\ &\leq \frac{D_\infty^2}{2}\|\sqrt{\widehat{\mathbf{v}}^{(t)}}\|_1 + \frac{1}{2(1-\beta_1)^2}\sum_{k=1}^{t}\alpha_k^2\|\mathbf{m}^{(k)}\|^2_{(\widehat{\mathbf{v}}^{(k)})^{-\frac{1}{2}}}.\end{aligned} \tag{3.1}$$

Lemma 2 *Let* $\{(\mathbf{x}^{(k)}, \mathbf{m}^{(k)}, \widehat{\mathbf{v}}^{(k)})\}$ *be the sequence from Algorithm 1. Under Assumption 2, it holds*

$$\mathbb{E}\|\mathbf{m}^{(k)}\|^2_{(\widehat{\mathbf{v}}^{(k)})^{-\frac{1}{2}}} \leq \frac{G_1}{\sqrt{1-\beta_2}}. \tag{3.2}$$

3.1 Convergence rate result for the case without delay

In this subsection, we use the previous two lemmas to show the convergence rate for the no-delay case, i.e., $\widehat{\mathbf{x}}^{(k)} = \mathbf{x}^{(k)}, \forall\, k \geq 1$ in (2.1). Although the no-delay case is not our main focus, our results improve over existing ones about AMSGrad.

Theorem 1 (Convex case without delay) *Let* $\{\mathbf{x}^{(k)}\}$ *be the sequence from Algorithm 1 with step size sequence* $\{\alpha_k\}$. *Given an integer* $K > 0$, *let* $\bar{\mathbf{x}}^{(K)} = \sum_{k=1}^{K}\frac{\sum_{j=k}^{K}\alpha_j\beta_1^{j-k}\mathbf{x}^{(k)}}{\sum_{t=1}^{K}\left(\sum_{j=t}^{K}\alpha_j\beta_1^{j-t}\right)}$. *Then under Assumptions 1–4, we have the following results:*

1. *If* $\alpha_k = \frac{\alpha}{\sqrt{K}}$, $\forall k \geq 1$, *for some* $\alpha > 0$, *then*

$$\mathbb{E}\left[F(\bar{\mathbf{x}}^{(K)}) - F^*\right] \leq \frac{nD_\infty^2 G_\infty + \frac{\alpha^2}{(1-\beta_1)^2}\frac{G_1}{\sqrt{1-\beta_2}}}{2\alpha\sqrt{K}(1-\beta_1)}. \tag{3.3}$$

2. *If* $\alpha_k = \frac{\alpha}{\sqrt{k}}$, $\forall k \geq 1$, *for some* $\alpha > 0$, *then*

$$\mathbb{E}\left[F(\bar{\mathbf{x}}^{(K)}) - F^*\right] \leq \frac{nD_\infty^2 G_\infty + \frac{\alpha^2(1+\log K)}{(1-\beta_1)^2}\frac{G_1}{\sqrt{1-\beta_2}}}{4\alpha(\sqrt{K+1}-1)(1-\beta_1)}. \tag{3.4}$$

Proof Taking expectation over both sides of (3.1) and using Lemma 2, we have

$$\begin{aligned}&(1-\beta_1)\sum_{k=1}^{t}\left(\sum_{j=k}^{t}\alpha_j\beta_1^{j-k}\right)\mathbb{E}\left\langle \mathbf{x}^{(k)}-\mathbf{x}, \mathbf{g}^{(k)}\right\rangle \\ &\leq \frac{D_\infty^2}{2}\mathbb{E}\|\sqrt{\widehat{\mathbf{v}}^{(t)}}\|_1 + \frac{G_1}{2(1-\beta_1)^2\sqrt{1-\beta_2}}\sum_{k=1}^{t}\alpha_k^2.\end{aligned} \tag{3.5}$$

From Assumption 4, we have

$$\mathbb{E}\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{g}^{(k)} \rangle = \mathbb{E}\langle \mathbf{x}^{(k)} - \mathbf{x}, \tilde{\nabla} F(\widehat{\mathbf{x}}^{(k)}) \rangle, \tag{3.6}$$

where $\tilde{\nabla} F(\widehat{\mathbf{x}}^{(k)}) \in \partial F(\widehat{\mathbf{x}}^{(k)})$. Hence, by the convexity of F and $\widehat{\mathbf{x}}^{(k)} = \mathbf{x}^{(k)}$, it holds $\mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big] \le \mathbb{E}\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{g}^{(k)} \rangle$, and thus (3.5) indicates

$$\begin{aligned} &(1-\beta_1) \sum_{k=1}^{t} \left(\sum_{j=k}^{t} \alpha_j \beta_1^{j-k} \right) \mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big] \\ &\le \frac{D_\infty^2}{2} \mathbb{E} \| \sqrt{\widehat{\mathbf{v}}^{(t)}} \|_1 + \frac{G_1}{2(1-\beta_1)^2 \sqrt{1-\beta_2}} \sum_{k=1}^{t} \alpha_k^2. \end{aligned} \tag{3.7}$$

Notice that $\sum_{j=k}^{K} \beta_1^{j-k} = \frac{1-\beta_1^{K-k+1}}{1-\beta_1} \in [1, \frac{1}{1-\beta_1})$. Hence, when $\alpha_k = \frac{\alpha}{\sqrt{K}}$, it holds $\sum_{k=1}^{K} \alpha_k^2 = \alpha^2$ and $\sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \ge \alpha \sqrt{K}$. By the convexity of F, we have

$$\sum_{t=1}^{K} \left(\sum_{j=t}^{K} \alpha_j \beta_1^{j-t} \right) F(\bar{\mathbf{x}}^{(K)}) \le \sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) F(\mathbf{x}^{(k)}). \tag{3.8}$$

In addition, we have from (7.6) and Assumption 3 that $\mathbb{E} \| \sqrt{\widehat{\mathbf{v}}^{(t)}} \|_1 \le n G_\infty$. Now divide both sides of (3.7) by $\sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right)$, take $\mathbf{x}$ as an optimal solution $\mathbf{x}^*$, and let $t = K$. We obtain the desired result in (3.3).

When $\alpha_k = \frac{\alpha}{\sqrt{k}}$, it holds

$$\sum_{k=1}^{K} \alpha_k^2 = \sum_{k=1}^{K} \frac{\alpha^2}{k} \le \alpha^2 + \int_1^K \frac{\alpha^2}{x} dx \le \alpha^2 (1 + \log K)$$

and

$$\sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \ge \sum_{k=1}^{K} \frac{\alpha}{\sqrt{k}} \ge \alpha \int_1^{K+1} \frac{\alpha}{\sqrt{x}} dx \ge 2\alpha(\sqrt{K+1} - 1).$$

Now utilizing the above bounds and (3.8) and following the same arguments to show (3.3), we obtain the desired result in (3.4) and completes the proof. □

Remark 1 Our rate is in the same order as that in [36]. However, by new analysis, we do not require an exponential or harmonic decaying sequence $\beta_{1,k}$ in computing $\mathbf{m}$ vectors in (1.2). The decaying weight is required in [36] and also its follow-up works such as [6, 27]. Numerically, a fixed weight β_1 can give significantly better results, and indeed [36] uses $\beta_{1,k} = \beta_1, \forall k$ in all its experiments. The recent works [8, 46]

have also weakened the condition on decreasing $\beta_{1,k}$ for smooth non-convex cases. However, none of these works has dropped the assumption $\beta_1 \leq \sqrt{\beta_2}$ that is required by [36]. Our result does not need this condition.

3.2 Convergence rate result for the case with delay

In this subsection, we analyze the proposed algorithm when there is delay, i.e., $\tau_k > 0$ in (2.2). The delay naturally happens for asynchronous computing. It causes one main difficulty in bounding the expected objective error $\mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big]$ from using (3.6). That is because the difference $\tilde{\nabla} F(\widehat{\mathbf{x}}^{(k)}) - \tilde{\nabla} F(\mathbf{x}^{(k)})$ in general will not vanish as $\widehat{\mathbf{x}}^{(k)} \neq \mathbf{x}^{(k)}$ caused by the delay. Nevertheless, an ergodic sublinear convergence result can still be guaranteed under a few additional mild assumptions.

Assumption 5 (Smoothness) F is L-smooth, i.e., $\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$, $\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

Assumption 6 (Bounded staleness) There is a finite integer τ such that $\tau_k \leq \tau$ for all $k \geq 1$.

With the master-worker set-up, we can measure the delay at the master, by counting the number of updates that are performed between two stochastic gradients computed by the same worker. Hence, by discarding too staled stochastic gradient, we can bound the staleness. In practice, we usually do not need to track the staleness. As mentioned in [26, 33], the staleness is usually roughly equal to the number of processors, if all the processors have similar computing ability.

The next theorem gives a generic result for the case with delay.

Theorem 2 *Let $\{\mathbf{x}^{(k)}\}$ be generated from Algorithm 1. Under Assumptions 1, 2, 4 and 5, we have that for any $\mathbf{x} \in X$,*

$$\begin{aligned}(1-\beta_1)\sum_{k=1}^{t}\left(\sum_{j=k}^{t}\alpha_j\beta_1^{j-k}\right)\mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big] \\ \leq \frac{D_\infty^2}{2}\mathbb{E}\|\sqrt{\widehat{\mathbf{v}}^{(t)}}\|_1 + \frac{1}{2(1-\beta_1)^2}\frac{G_1}{\sqrt{1-\beta_2}}\sum_{k=1}^{t}\alpha_k^2 \\ + \frac{L(1-\beta_1)}{2}\sum_{k=1}^{t}\left(\sum_{j=k}^{t}\alpha_j\beta_1^{j-k}\right)\mathbb{E}\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2.\end{aligned} \tag{3.9}$$

Proof It follows from the convexity of F that

$$\big\langle \widehat{\mathbf{x}}^{(k)} - \mathbf{x}, \nabla F(\widehat{\mathbf{x}}^{(k)})\big\rangle \geq F(\widehat{\mathbf{x}}^{(k)}) - F(\mathbf{x}),$$

and in addition, the L-smoothness of F implies

$$\big\langle \mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}, \nabla F(\widehat{\mathbf{x}}^{(k)})\big\rangle \geq F(\mathbf{x}^{(k)}) - F(\widehat{\mathbf{x}}^{(k)}) - \frac{L}{2}\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2.$$

Plugging the above two inequalities into (3.6), we have

$$\mathbb{E}\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{g}^{(k)} \rangle \geq \mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big] - \frac{L}{2}\mathbb{E}\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2,$$

which together with (3.5) gives the desired result in (3.9). □

Note that if $\mathbf{x}^{(k)} = \widehat{\mathbf{x}}^{(k)}, \forall\, k$, the inequality in (3.9) reduces to that in (3.7). However, due to the asynchrony, we generally only have the relation in (2.2). Therefore, the term about $\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2$ in (3.9) is not zero, and we need to bound it appropriately in order to establish the sublinear convergence. From (2.4), it suffices to bound $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|^2$ for all k, which can be obtained by the following lemmas. The proofs of the lemmas are given in the appendix.

Lemma 3 (Non-expansiveness) *Suppose $X = [a_1, b_1] \times \cdots \times [a_n, b_n]$ for some finite numbers $\{a_i\}$ and $\{b_i\}$. Let $\{\mathbf{x}^{(k)}\}$ be generated from Algorithm 1, and for any $i \in [n]$, we let $x_i^{(k+1)} = x_i^{(k)}$, if $\widehat{v}_i^{(k)} = 0$. Then for any $k \geq 1$, it holds*

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \alpha_k \big\|\mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\big\|. \tag{3.10}$$

Remark 2 Notice that when $\widehat{v}_i^{(k)} = 0$, we must have $m_i^{(k)} = 0$ and in this case, x_i does not affect the objective of (1.5). Hence, when X is separable and $\widehat{v}_i^{(k)} = 0$, $x_i^{(k+1)} = x_i^{(k)}$ is one optimal choice for x_i, and thus such a setting will not affect the optimality condition of (1.5).

Lemma 4 *Let $\{(\mathbf{g}^{(k)}, \mathbf{m}^{(k)}, \widehat{\mathbf{v}}^{(k)})\}$ be generated from Algorithm 1. It holds for any $k \geq 1$ that*

$$\|\mathbf{g}^{(j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\| \leq \frac{\sqrt{\|\mathbf{g}^{(j)}\|_0}}{\sqrt{1-\beta_2}}, \forall\, j \leq k, \tag{3.11a}$$

$$\|\mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\| \leq \sum_{j=1}^{k} (1-\beta_1)\beta_1^{k-j} \frac{\sqrt{\|\mathbf{g}^{(j)}\|_0}}{\sqrt{1-\beta_2}}, \tag{3.11b}$$

$$\|\mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\|^2 \leq \frac{1-\beta_1}{1-\beta_2} \sum_{j=1}^{k} \beta_1^{k-j} \|\mathbf{g}^{(j)}\|_0. \tag{3.11c}$$

Applying the results in the above two lemmas to the inequality in (3.9), we establish the sublinear convergence result of Algorithm 1 as follows.

Theorem 3 (convex case with delay) *Suppose Assumptions 1 through 6 hold. Assume $X = [a_1, b_1] \times \cdots \times [a_n, b_n]$ for finite numbers $\{a_i\}$ and $\{b_i\}$. Let $\{\mathbf{x}^{(k)}\}$ be generated from Algorithm 1, and for any $i \in [n]$, we let $x_i^{(k+1)} = x_i^{(k)}$, if $\widehat{v}_i^{(k)} = 0$. Given a positive integer K and $\alpha > 0$, let $\alpha_k = \frac{\alpha}{\sqrt{K}}$ for all $1 \leq k \leq K$. Then*

$$\mathbb{E}\big[F(\bar{\mathbf{x}}^{(K)}) - F^*\big] \leq \frac{1}{2\alpha\sqrt{K}(1-\beta_1)} \left(n D_\infty^2 G_\infty + \frac{\alpha^2 G_1}{(1-\beta_1)^2\sqrt{1-\beta_2}} + \frac{\alpha^3 L \tau^2 n}{\sqrt{K}(1-\beta_2)} \right), \tag{3.12}$$

where $\bar{\mathbf{x}}^{(K)}$ *is drawn from* $\{\mathbf{x}^{(k)}\}_{k=1}^{K}$ *with*

$$\text{Prob}(\bar{\mathbf{x}}^{(K)} = \mathbf{x}^{(k)}) = \frac{\sum_{j=k}^{K} \alpha_j \beta_1^{j-k}}{\sum_{t=1}^{K} \left(\sum_{j=t}^{K} \alpha_j \beta_1^{j-t} \right)}, \forall\, 1 \le k \le K.$$

Proof By (3.10) and (3.11c), we have $\mathbb{E}\|\mathbf{x}^{(k-l+1)} - \mathbf{x}^{(k-l)}\|^2 \le \frac{n\alpha_{k-l}^2}{1-\beta_2}$. Since $\tau_k \le \tau, \forall\, k \ge 1$, it follows from (2.4) that

$$\mathbb{E}\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2 \le \tau \sum_{l=1}^{\tau} \mathbb{E}\|\mathbf{x}^{(k-l+1)} - \mathbf{x}^{(k-l)}\|^2.$$

In addition, notice that

$$\sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \left(\tau \sum_{l=1}^{\tau} \alpha_{k-l}^2 \right) \le \frac{\tau^2 \alpha^3}{(1-\beta_1)\sqrt{K}}.$$

Therefore

$$\frac{L(1-\beta_1)}{2} \sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \mathbb{E}\|\mathbf{x}^{(k)} - \widehat{\mathbf{x}}^{(k)}\|^2 \le \frac{\alpha^3 L \tau^2 n}{2\sqrt{K}(1-\beta_2)}.$$

Plugging the above inequality into (3.9) with $t = K$ and using $\sum_{k=1}^{K} \alpha_k^2 = \alpha^2$ give

$$\begin{aligned} &(1-\beta_1) \sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \mathbb{E}\big[F(\mathbf{x}^{(k)}) - F(\mathbf{x})\big] \\ &\le \frac{D_\infty^2}{2} \mathbb{E}\|\sqrt{\widehat{\mathbf{v}}^{(t)}}\|_1 + \frac{\alpha^2}{2(1-\beta_1)^2} \frac{G_1}{\sqrt{1-\beta_2}} + \frac{\alpha^3 L \tau^2 n}{2\sqrt{K}(1-\beta_2)}. \end{aligned}$$

Now using the definition of $\bar{\mathbf{x}}^{(K)}$ and noting $\sum_{k=1}^{K} \left(\sum_{j=k}^{K} \alpha_j \beta_1^{j-k} \right) \ge \alpha\sqrt{K}$, we obtain the result in (3.12) and complete the proof. □

Remark 3 (How delay affects convergence speed) Take $\alpha = O(1)$. Then (3.12) implies that the effect by the delay decreases at the rate of $K^{\frac{1}{4}}$, namely, we can achieve nearly-linear speed-up if $\tau = o(K^{\frac{1}{4}})$. Peng et al. [33] shows that the delay, in expectation, equals the number of processors if all of them have the same computing power. Hence, in the ideal case, we can expect nearly-linear speed-up by using $o(K^{\frac{1}{4}})$ processors. However, notice that the convergence result of the asynchronous case requires stronger assumptions than that of the synchronous counterpart. In addition, we set $\alpha_k = \frac{\alpha}{\sqrt{K}}$ for all $1 \le k \le K$ in Theorem 3 for simplicity. A sublinear convergence result can also be established if $\alpha_k = \frac{\alpha}{\sqrt{k}}, \forall\, k$ by similar arguments as those in the proof of Theorem 1.

4 Convergence for non-convex problems

In this section, we analyze Algorithm 1 for non-convex problems under Assumptions 3–6. Due to the difficulty caused by nonconvexity, we assume $X = \mathbb{R}^n$. Then the update in (1.5) becomes

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}. \tag{4.1}$$

When $X = \mathbb{R}^n$, the gradient boundedness condition in Assumption 3 may not hold for deep learning problems. However, we are unable to relax this strong assumption. It is also made in all existing works that analyze adaptive SGMs for non-convex problems, e.g., [8, 46]. On analyzing nonadaptive SGMs for non-convex problems, this assumption can be relaxed to a variance boundedness condition [15, 23], which may not hold either for unconstrained deep learning problems but is weaker than the gradient boundedness condition.

Given a maximum number K of iterations, we will assume, without loss of generality, $\widehat{v}_i^{(K)} > 0$ for all $i \in [n]$. Note that if $\widehat{v}_i^{(K)} = 0$ for some i, then $g_i^{(k)} = 0$ for all $k \le K$, and in this case, x_i never changes and can be simply viewed as a constant instead of a variable. We define an auxiliary sequence $\{\widetilde{\mathbf{v}}^{(k)}\}_{k=1}^K$ and gradient bounds as follows. These are only used in our analysis but not in the computation.

Definition 1 Given a positive integer K, let $\{\widehat{\mathbf{v}}^{(k)}\}_{k=1}^K$ be computed from Algorithm 1. For any $i \in [n]$, suppose $k_i \le K$ is the smallest number such that $\widehat{v}_i^{(k_i)} > 0$. Define $\{\widetilde{\mathbf{v}}^{(k)}\}_{k=1}^K$ as $\widetilde{v}_i^{(k)} = \max\{\widehat{v}_i^{(k)}, \widehat{v}_i^{(k_i)}\}$ for all $i \in [n]$ and all $k \in [K]$. Denote $\widetilde{\mathbf{V}}^{(k)} = \mathrm{Diag}(\widetilde{\mathbf{v}}^{(k)})$ for all $k \in [K]$.

Definition 2 Given a positive integer K, let $\{\mathbf{g}^{(k)}\}_{k=1}^K$ and $\{\mathbf{x}^{(k)}\}_{k=1}^K$ be computed as in Algorithm 1. We define $(\boldsymbol{\Gamma}(K), \boldsymbol{\Phi}(K))$ as:

$$\Gamma_i(K) = \max_{1\le k\le K} |g_i^{(k)}|, \text{ and } \Phi_i(K) = \max_{1\le k\le K} |\nabla_i F(\mathbf{x}^{(k)})|, \ \forall\, i \in [n]. \tag{4.2}$$

We abbreviate the pair as $(\boldsymbol{\Gamma}, \boldsymbol{\Phi})$ to hide the dependence on K, when it is clear from the context.

Remark 4 We make a few remarks on $\{\widetilde{\mathbf{v}}^{(k)}\}_{k=1}^K$. (I) Assume $\widehat{v}_i^{(K)} > 0$ for all $i \in [n]$. Then each $\widetilde{\mathbf{v}}^{(k)}$ is a positive vector, and $\widetilde{\mathbf{v}}^{(k)} \ge \widetilde{\mathbf{v}}^{(k-1)}$ still holds component-wisely; (II) $\mathbf{m}^{(k)} \oslash \sqrt{\widetilde{\mathbf{v}}^{(k)}} = \mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}$ and $\mathbf{g}^{(k)} \oslash \sqrt{\widetilde{\mathbf{v}}^{(k)}} = \mathbf{g}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}$ for all $k \le K$; and (III) $\widetilde{\mathbf{v}}^{(K)} = \widehat{\mathbf{v}}^{(K)}$ under the assumption $\widehat{\mathbf{v}}^{(K)} > \mathbf{0}$.

4.1 Preparatory lemmas

In this subsection, we establish several lemmas. Their proofs are given in the appendix. The next lemma gives bounds on Γ_i and Φ_i under Assumption 3.

Lemma 5 *Given a positive integer K, let $\{\widehat{\mathbf{v}}^{(k)}\}_{k=1}^K$ and $\{\mathbf{m}^{(k)}\}_{k=1}^K$ be generated from Algorithm 1, and let $(\boldsymbol{\Gamma}, \boldsymbol{\Phi})$ be given in Definition 2. We have for all $i \in [n]$, $|m_i^{(k)}| \le$*

Γ_i, *and* $\widehat{v}_i^{(k)} \leq \Gamma_i^2$. *Moreover, if Assumption 3 holds, then* $\Gamma_i \leq G_\infty$ *almost surely, and* $\Phi_i \leq G_\infty$ *for all* $i \in [n]$.

To analyze Algorithm 1 for non-convex problems, we follow the analytical framework of [44]. Let $\mathbf{x}^{(0)} = \mathbf{x}^{(1)}$, and we define an auxiliary sequence $\mathbf{z}^{(k)}$ as follows:

$$\mathbf{z}^{(k)} = \mathbf{x}^{(k)} + \frac{\beta_1}{1-\beta_1}(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = \frac{1}{1-\beta_1}\mathbf{x}^{(k)} - \frac{\beta_1}{1-\beta_1}\mathbf{x}^{(k-1)}, \ \forall k \geq 1. \quad (4.3)$$

The following lemma is from Lemma A.3 of [46]. It shows that $\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}$ can be represented in two different ways. However, due to typos in the original proof, we provide a complete proof in the appendix for the convenience of the readers.

Lemma 6 *Let* $\mathbf{z}^{(k)}$ *be defined as in* (4.3) *and* $\widetilde{\mathbf{V}}^{(k)}$ *in Definition 1. We have*

$$\mathbf{z}^{(2)} - \mathbf{z}^{(1)} = -\alpha_1(\widetilde{\mathbf{V}}^{(1)})^{-\frac{1}{2}}\mathbf{g}^{(1)}, \quad (4.4)$$

and for $k = 2, \ldots, K$,

$$\begin{aligned}
&\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)} \\
&= \frac{\beta_1}{1-\beta_1}\left[\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\right]\mathbf{m}^{(k-1)} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}, \quad (4.5)\\
&= \frac{\beta_1}{1-\beta_1}\left[\mathbf{I} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\alpha_{k-1}^{-1}(\widetilde{\mathbf{V}}^{(k-1)})^{\frac{1}{2}}\right](\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}) - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}. \quad (4.6)
\end{aligned}$$

Lemma 7 *Let* $\{\mathbf{x}^{(k)}\}$ *be from Algorithm 1,* $\{\mathbf{z}^{(k)}\}$ *defined as in* (4.3), *and* $\{\widetilde{\mathbf{v}}^{(k)}\}$ *in Definition 1. Also, let* $\{\alpha_k\}$ *be a non-increasing positive sequence. For* $k = 2, \ldots, K$, *we have*

$$\begin{aligned}
\nabla F(\mathbf{x}^{(k)})^\top(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) \leq &-\nabla F(\mathbf{x}^{(k)})^\top\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{g}^{(k)} \\
&+ \frac{1}{1-\beta_1}\sum_{i=1}^n \Gamma_i\Phi_i\left[\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right]. \quad (4.7)
\end{aligned}$$

The next two lemmas are directly from [46]. Although the original results are for $k \geq 2$, they trivially hold when $k = 1$.

Lemma 8 (Lemma A.4 of [46]) *Let* $\{\mathbf{z}^{(k)}\}$ *be defined as in* (4.3), *and let* $\{\alpha_k\}_{k\geq 1}$ *be a non-increasing positive sequence. For* $k \geq 1$, *we have*

$$\|\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\| \leq \frac{\beta_1}{1-\beta_1}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\| + \|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|. \quad (4.8)$$

Lemma 9 (Lemma A.5 of [46]) *Let* $\{\mathbf{z}^{(k)}\}$ *be defined as in* (4.3) *and* $\widetilde{\mathbf{V}}^{(k)}$ *in Definition* 1. *Under Assumption* 5, *we have*

$$\|\nabla F(\mathbf{z}^{(k)}) - \nabla F(\mathbf{x}^{(k)})\| \leq \frac{L\beta_1}{1-\beta_1}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\|, \forall\, k \geq 1. \tag{4.9}$$

We still need the following lemma to show our main convergence result for the non-convex case.

Lemma 10 *Let* $\{\mathbf{z}^{(k)}\}$ *be defined as in* (4.3) *and* $\widetilde{\mathbf{V}}^{(k)}$ *in Definition* 1. *Also, let* $\{\alpha_k\}$ *be a non-increasing positive sequence. Under Assumption* 5, *we have that for all* $k \geq 1$,

$$\begin{aligned}&\left(\nabla F(\mathbf{z}^{(k)}) - \nabla F(\mathbf{x}^{(k)})\right)^\top (\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)})\\ &\leq \frac{3L\beta_1^2}{2(1-\beta_1)^2}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\|^2 + \frac{L}{2}\|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|^2,\end{aligned} \tag{4.10}$$

and

$$\|\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\|^2 \leq \frac{4\beta_1^2}{(1-\beta_1)^2}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\|^2 + \frac{4}{3}\|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|^2. \tag{4.11}$$

4.2 Convergence rate results

By the lemmas established in the previous subsection, we are ready to show the convergence result of Algorithm 1 for non-convex problems. The next theorem gives the convergence rate without specifying the learning rate $\{\alpha_k\}$.

Theorem 4 *Given an integer* $K \geq 2$, *let* $\{\mathbf{x}^{(k)}\}_{k=1}^K$ *and* $\{\widehat{\mathbf{v}}^{(k)}\}_{k=1}^K$ *be generated from Algorithm* 1 *with a non-increasing positive sequence* $\{\alpha_k\}_{k=1}^K$. *Suppose* $\widehat{\mathbf{v}}^{(K)} > \mathbf{0}$. *Suppose that there is a constant* C_F *such that* $|F(\mathbf{x})| \leq C_F$, $\forall\, \mathbf{x}$. *Let* $2 \leq k_0 \leq K$ *and* $\bar{\mathbf{x}}^{(k_0,K)}$ *be drawn from* $\{\mathbf{x}^{(k)}\}_{k=k_0}^K$ *with probability*

$$\text{Prob}(\bar{\mathbf{x}}^{(k_0,K)} = \mathbf{x}^{(k)}) = \frac{\alpha_{k-1}}{\sum_{j=k_0}^K \alpha_{j-1}},\ \forall\, k = k_0, \ldots, K. \tag{4.12}$$

Then under Assumptions 3 *through* 6, *it holds*

$$\begin{aligned}&\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2\\ &\leq \frac{G_\infty^3 \mathbb{E}\|(\widehat{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1}\frac{\alpha_{k_0-1}}{\sum_{k=k_0}^K \alpha_{k-1}} + \frac{2C_F G_\infty}{\sum_{k=k_0}^K \alpha_{k-1}}\\ &\quad + \frac{7nLG_\infty}{6(1-\beta_2)}\frac{\sum_{k=k_0}^K \alpha_k^2}{\sum_{k=k_0}^K \alpha_{k-1}} + \frac{7nLG_\infty\beta_1^2}{2(1-\beta_2)(1-\beta_1)^2}\cdot\frac{\sum_{k=k_0}^K \alpha_{k-1}^2}{\sum_{k=k_0}^K \alpha_{k-1}}\\ &\quad + \frac{\sqrt{n}LG_\infty \sum_{k=k_0}^K \alpha_{k-1}\sum_{j=1}^{\tau}\alpha_{k-j}\sqrt{\mathbb{E}\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\|^2}}{\sqrt{1-\beta_2}\sum_{t=k_0}^K \alpha_{t-1}}.\end{aligned} \tag{4.13}$$

Proof By the L-smoothness of F, it follows that

$$\begin{aligned} F(\mathbf{z}^{(k+1)}) &\le F(\mathbf{z}^{(k)}) + \nabla F(\mathbf{z}^{(k)})^\top(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) + \tfrac{L}{2}\|\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\|^2 \\ &= F(\mathbf{z}^{(k)}) + \nabla F(\mathbf{x}^{(k)})^\top(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) \\ &\quad + \left(\nabla F(\mathbf{z}^{(k)}) - \nabla F(\mathbf{x}^{(k)})\right)^\top(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) + \tfrac{L}{2}\|\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\|^2. \end{aligned} \tag{4.14}$$

For $2 \le k \le K$, we substitute (4.7), (4.10) and (4.11) into (4.14) and rearrange terms to have

$$\begin{aligned} &F(\mathbf{z}^{(k+1)}) + \tfrac{\sum_{i=1}^n \Gamma_i \Phi_i \alpha_k (\widetilde{v}_i^{(k)})^{-\frac{1}{2}}}{1-\beta_1} - F(\mathbf{z}^{(k)}) - \tfrac{\sum_{i=1}^n \Gamma_i \Phi_i \alpha_{k-1} (\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}}}{1-\beta_1} \\ &\le -\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1} (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \mathbf{g}^{(k)} + \tfrac{7L\beta_1^2}{2(1-\beta_1)^2}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\|^2 \\ &\quad + \tfrac{7L}{6}\|\alpha_k (\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}} \mathbf{g}^{(k)}\|^2 \\ &= -\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1} (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \mathbf{g}^{(k)} + \tfrac{7L\beta_1^2}{2(1-\beta_1)^2}\|\alpha_{k-1}\mathbf{m}^{(k-1)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|^2 \\ &\quad + \tfrac{7L}{6}\|\alpha_k (\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}} \mathbf{g}^{(k)}\|^2. \end{aligned} \tag{4.15}$$

From Assumption 4, we have $\mathbb{E}\left[\mathbf{g}^{(k)} \,|\, \mathcal{H}_k\right] = \nabla F(\widehat{\mathbf{x}}^{(k)})$, and thus taking the expectation on both sides of (4.15) gives

$$\begin{aligned} &\mathbb{E}\left[F(\mathbf{z}^{(k+1)}) + \tfrac{G_\infty^2 \alpha_k \|(\widetilde{\mathbf{v}}^{(k)})^{-\frac{1}{2}}\|_1}{1-\beta_1} - F(\mathbf{z}^{(k)}) - \tfrac{G_\infty^2 \alpha_{k-1} \|(\widetilde{\mathbf{v}}^{(k-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1}\right] \\ &\le \mathbb{E}\left[F(\mathbf{z}^{(k+1)}) + \tfrac{\sum_{i=1}^n \Gamma_i \Phi_i \alpha_k (\widetilde{v}_i^{(k)})^{-\frac{1}{2}}}{1-\beta_1} - F(\mathbf{z}^{(k)}) - \tfrac{\sum_{i=1}^n \Gamma_i \Phi_i \alpha_{k-1} (\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}}}{1-\beta_1}\right] \\ &\le \mathbb{E}\Bigg[\tfrac{7L}{6}\|\alpha_k (\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}} \mathbf{g}^{(k)}\|^2 - \nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1} (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\widehat{\mathbf{x}}^{(k)}) \\ &\quad + \tfrac{7L\beta_1^2}{2(1-\beta_1)^2}\|\alpha_{k-1}\mathbf{m}^{(k-1)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|^2\Bigg] \\ &= \mathbb{E}\left[\tfrac{7L}{6}\|\alpha_k (\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}} \mathbf{g}^{(k)}\|^2 + \tfrac{7L\beta_1^2}{2(1-\beta_1)^2}\|\alpha_{k-1}\mathbf{m}^{(k-1)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|^2\right] \\ &\quad + \mathbb{E}\left[\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1} (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \left(\nabla F(\mathbf{x}^{(k)}) - \nabla F(\widehat{\mathbf{x}}^{(k)})\right)\right] \\ &\quad - \mathbb{E}\left[\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1} (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\mathbf{x}^{(k)})\right], \end{aligned} \tag{4.16}$$

where the first inequality follows from $\alpha_k (\widetilde{v}_i^{(k)})^{-\frac{1}{2}} - \alpha_{k-1} (\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} \le 0$ for all $i \in [n]$ and Lemma 5. By the Cauchy–Schwarz inequality, the smoothness of F, Assumption 6, and Eqs. (2.3) and (4.1), we have

$$\nabla F(\mathbf{x}^{(k)})^\top (\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \left(\nabla F(\mathbf{x}^{(k)}) - \nabla F(\widehat{\mathbf{x}}^{(k)})\right)$$

$$\leq \|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\| \cdot \|\nabla F(\mathbf{x}^{(k)}) - \nabla F(\widehat{\mathbf{x}}^{(k)})\|$$
$$\leq L\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\| \left(\textstyle\sum_{j=1}^{\tau} \|\alpha_{k-j}\mathbf{m}^{(k-j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-j)}}\|\right).$$

In addition, by Definition 1 and Lemma 5, it follows

$$\mathbb{E}\left[\nabla F(\mathbf{x}^{(k)})^{\top}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\right]$$
$$\geq \mathbb{E}\left[\|\widetilde{\mathbf{v}}^{(k-1)}\|_{\infty}^{-\frac{1}{2}}\|\nabla F(\mathbf{x}^{(k)})\|^2\right] \geq G_{\infty}^{-1}\mathbb{E}\|\nabla F(\mathbf{x}^{(k)})\|^2.$$

Substituting the above two inequalities into (4.16), we have

$$\mathbb{E}\left[F(\mathbf{z}^{(k+1)}) + \frac{G_{\infty}^2\alpha_k\|(\widetilde{\mathbf{v}}^{(k)})^{-\frac{1}{2}}\|_1}{1-\beta_1} - F(\mathbf{z}^{(k)}) - \frac{G_{\infty}^2\alpha_{k-1}\|(\widetilde{\mathbf{v}}_i^{(k-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1}\right]$$
$$\leq \mathbb{E}\left[\frac{7L}{6}\|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|^2 + \frac{7L\beta_1^2}{2(1-\beta_1)^2}\|\alpha_{k-1}\mathbf{m}^{(k-1)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|^2\right]$$
$$+ \mathbb{E}\left[\alpha_{k-1}L\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\| \left(\textstyle\sum_{j=1}^{\tau} \|\alpha_{k-j}\mathbf{m}^{(k-j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-j)}}\|\right)\right]$$
$$+ \mathbb{E}\left[-\frac{\alpha_{k-1}}{G_{\infty}}\|\nabla F(\mathbf{x}^{(k)})\|^2\right]. \tag{4.17}$$

For any $2 \leq k_0 \leq K$, summing (4.17) over $k = k_0, \ldots, K$ and using the condition $|F(\mathbf{x})| \leq C_F$, $\forall\, \mathbf{x}$, we have

$$G_{\infty}^{-1}\textstyle\sum_{k=k_0}^{K}\alpha_{k-1}\mathbb{E}\|\nabla F(\mathbf{x}^{(k)})\|^2$$
$$\leq 2C_F + \frac{G_{\infty}^2\alpha_{k_0-1}\mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1} + \frac{7L}{6}\textstyle\sum_{k=k_0}^{K}\mathbb{E}\|\alpha_k\mathbf{g}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\|^2$$
$$+ \frac{7L\beta_1^2}{2(1-\beta_1)^2}\textstyle\sum_{k=k_0}^{K}\mathbb{E}\|\alpha_{k-1}\mathbf{m}^{(k-1)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|^2$$
$$+ L\sum_{k=k_0}^{K}\alpha_{k-1}\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\|\left(\sum_{j=1}^{\tau}\|\alpha_{k-j}\mathbf{m}^{(k-j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k-j)}}\|\right)\right]. \tag{4.18}$$

Now use (3.11b) of Lemma 4 in the above inequality to have

$$G_{\infty}^{-1}\textstyle\sum_{k=k_0}^{K}\alpha_{k-1}\mathbb{E}\|\nabla F(\mathbf{x}^{(k)})\|^2$$
$$\leq 2C_F + \frac{G_{\infty}^2\alpha_{k_0-1}\mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1}$$
$$+ \frac{7nL}{6(1-\beta_2)}\textstyle\sum_{k=k_0}^{K}\alpha_k^2 + \frac{7nL\beta_1^2}{2(1-\beta_2)(1-\beta_1)^2}\sum_{k=k_0}^{K}\alpha_{k-1}^2$$
$$+ \frac{L}{\sqrt{1-\beta_2}}\textstyle\sum_{k=k_0}^{K}\alpha_{k-1}\sum_{j=1}^{\tau}\alpha_{k-j}\sum_{l=1}^{k-j}(1-\beta_1)\beta_1^{k-j-l}$$
$$\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\|\sqrt{\|\mathbf{g}^{(l)}\|_0}\right]$$

$$\leq 2C_F + \frac{G_\infty^2 \alpha_{k_0-1} \mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{1-\beta_1} + \frac{7nL}{6(1-\beta_2)} \sum_{k=k_0}^K \alpha_k^2 + \frac{7nL\beta_1^2}{2(1-\beta_2)(1-\beta_1)^2} \sum_{k=k_0}^K \alpha_{k-1}^2 + \frac{\sqrt{n}L}{\sqrt{1-\beta_2}} \sum_{k=k_0}^K \alpha_{k-1} \sqrt{\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\mathbf{x}^{(k)})\|^2\right]} \sum_{j=1}^\tau \alpha_{k-j}, \quad (4.19)$$

where the last inequality is by Cauchy–Schwarz inequality. We obtain the desired result in (4.13) by dividing $G_\infty^{-1} \sum_{k=k_0}^K \alpha_{k-1}$ on both sides of (4.19) and using the definition of $\bar{\mathbf{x}}^{(k_0,K)}$. □

Below we specify the choices of $\{\alpha_k\}$ and show the sublinear convergence.

Theorem 5 *Given an integer $K \geq 2$, let $\{\mathbf{x}^{(k)}\}_{k=1}^K$ and $\{\widehat{\mathbf{v}}^{(k)}\}_{k=1}^K$ be generated from Algorithm 1 with a non-increasing positive sequence $\{\alpha_k\}_{k=1}^K$. For some $2 \leq k_0 \leq K$, let $\bar{\mathbf{x}}^{(k_0,K)}$ be drawn from $\{\mathbf{x}^{(k)}\}_{k=k_0}^K$ according to (4.12). Suppose Assumptions 3 through 6 hold. In addition, assume $\widehat{\mathbf{v}}^{(K)} > \mathbf{0}$. Moreover, suppose that there are positive constants C_F and c such that $|F(\mathbf{x})| \leq C_F$, $\forall \mathbf{x}$, and $\widetilde{v}_i^{(k_0-1)} \geq c^2$, $\forall i \in [n]$ hold almost surely. The following results hold:*

1. *If $\alpha_k = \frac{\alpha}{\sqrt{K-k_0+1}}$ for all k and some constant $\alpha > 0$, then*

$$\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2 \leq C_1 + \frac{C_2}{c}\left(\sqrt{C_1} + \frac{C_2}{c}\right), \quad (4.20)$$

where $C_2 = \frac{\alpha\tau\sqrt{n}LG_\infty}{\sqrt{1-\beta_2}\sqrt{K-k_0+1}}$, and

$$C_1 = \frac{G_\infty^3 \mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{(1-\beta_1)(K-k_0+1)} + \frac{2C_F G_\infty}{\alpha\sqrt{K-k_0+1}} + \frac{7nLG_\infty(1-2\beta_1+4\beta_1^2)}{6(1-\beta_2)(1-\beta_1)^2} \frac{\alpha}{\sqrt{K-k_0+1}}.$$

2. *Let $k_0 = \frac{K}{2} \geq \tau + 2$. If $\alpha_k = \frac{\alpha}{\sqrt{k}}$ for all $k \geq 1$, then (4.20) holds, where $C_2 = \frac{2\sqrt{2}\alpha\tau\sqrt{n}LG_\infty}{\sqrt{K}\sqrt{1-\beta_2}}$, and*

$$C_1 = \frac{G_\infty^3 \mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1}{(2-\sqrt{2})(1-\beta_1)\sqrt{K}\sqrt{K/2-1}} + \frac{2C_F G_\infty}{(2-\sqrt{2})\alpha\sqrt{K}} + \frac{7nLG_\infty}{6(1-\beta_2)} \frac{\alpha\log 4}{(2-\sqrt{2})\sqrt{K}} + \frac{7nLG_\infty\beta_1^2}{2(1-\beta_2)(1-\beta_1)^2} \frac{\alpha(1+\log 3)}{(2-\sqrt{2})\sqrt{K}}.$$

Proof **Setting 1:** When $\alpha_k = \frac{\alpha}{\sqrt{K-k_0+1}}$, $\forall 1 \leq k \leq K$, we plug α_k into (4.13) and have

$$\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2 \leq C_1 + \frac{C_2}{K-k_0+1} \sum_{k=k_0}^K \sqrt{\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\mathbf{x}^{(k)})\|^2\right]}. \quad (4.21)$$

Since $\widetilde{v}_i^{(k_0-1)} \geq c^2, \forall i \in [n]$ almost surely and $\widetilde{\mathbf{v}}^{(k+1)} \geq \widetilde{\mathbf{v}}^{(k)}, \forall k \geq 1$, it holds

$$\begin{aligned}&\frac{1}{K-k_0+1}\sum_{k=k_0}^{K}\sqrt{\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\nabla F(\mathbf{x}^{(k)})\|^2\right]}\\ \leq\ &\frac{1}{c(K-k_0+1)}\sum_{k=k_0}^{K}\sqrt{\mathbb{E}\|\nabla F(\mathbf{x}^{(k)})\|^2} \leq \frac{1}{c}\sqrt{\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2},\end{aligned}$$

where the last inequality follows from Jensen's inequality. Hence, plugging the above inequality into (4.21) yields

$$\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2 \leq C_1 + \frac{C_2}{c}\sqrt{\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2}, \tag{4.22}$$

which implies $\sqrt{\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2} \leq \sqrt{C_1}+\frac{C_2}{c}$. Therefore, we have the desired result from (4.22).

Setting 2 When $\alpha_k = \frac{\alpha}{\sqrt{k}}$, $\forall 1 \leq k \leq K$, we have

$$\begin{aligned}\sum_{k=k_0}^{K}\alpha_{k-1} &= \sum_{k=k_0}^{K}\frac{\alpha}{\sqrt{k-1}}\\ &\geq \int_{k_0-1}^{K}\frac{\alpha}{\sqrt{x}}dx = 2\alpha(\sqrt{K}-\sqrt{k_0-1}) \geq (2-\sqrt{2})\alpha\sqrt{K},\end{aligned}$$

and

$$\sum_{k=k_0}^{K}\alpha_k^2 = \sum_{k=k_0}^{K}\frac{\alpha^2}{k} \leq \int_{k_0-1}^{K}\frac{\alpha^2}{x}dx = \alpha^2\log\frac{K}{k_0-1} = \alpha^2\log\frac{1}{\frac{1}{2}-\frac{1}{K}} \leq \alpha^2\log 4.$$

Similarly,

$$\begin{aligned}\sum_{k=k_0}^{K}\alpha_{k-1}^2 &= \sum_{k=k_0}^{K}\frac{\alpha^2}{k-1} \leq \frac{\alpha^2}{k_0-1} + \int_{k_0-1}^{K-1}\frac{\alpha^2}{x}dx\\ &\leq\ \alpha^2 + \alpha^2\log\frac{K-1}{k_0-1} \leq \alpha^2(1+\log 3),\end{aligned}$$

and for all $k \geq k_0 \geq \tau+2$,

$$\begin{aligned}\sum_{j=1}^{\tau}\alpha_{k-j} &\leq \sum_{j=1}^{\tau}\alpha_{k_0-j} = \sum_{j=1}^{\tau}\frac{\alpha}{\sqrt{k_0-j}}\\ &\leq\ \int_{k_0-\tau-1}^{k_0-1}\frac{\alpha}{\sqrt{x}}dx = 2\alpha(\sqrt{k_0-1}-\sqrt{k_0-\tau-1}) \leq \frac{2\alpha\tau}{\sqrt{k_0}}.\end{aligned}$$

Therefore, plugging $\alpha_k = \frac{\alpha}{\sqrt{k}}$, $\forall 1 \le k \le K$ into (4.13) and using the above inequalities, we have

$$\begin{aligned}
&\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2 \\
&\le C_1 + \frac{C_2}{\sum_{k=k_0}^K \alpha_{k-1}} \sum_{k=k_0}^K \alpha_{k-1} \sqrt{\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\mathbf{x}^{(k)})\|^2\right]}.
\end{aligned} \tag{4.23}$$

Notice $\widetilde{v}_i^{(k_0-1)} \ge c^2$, $\forall i \in [n]$ almost surely, and also use the definition of $\bar{\mathbf{x}}^{(k_0,K)}$ in (4.12). We have, by Jensen's inequality,

$$\begin{aligned}
&\frac{1}{\sum_{k=k_0}^K \alpha_{k-1}} \sum_{k=k_0}^K \alpha_{k-1} \sqrt{\mathbb{E}\left[\|(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} \nabla F(\mathbf{x}^{(k)})\|^2\right]} \\
&\le \frac{1}{c} \sqrt{\mathbb{E}\|\nabla F(\bar{\mathbf{x}}^{(k_0,K)})\|^2}.
\end{aligned}$$

Now by the same arguments as those in the proof of Setting 1, we obtain the desired result. □

Remark 5 We make a few remarks here about Theorem 5. (i) Note that $\mathbb{E}\|(\widetilde{\mathbf{v}}^{(k_0-1)})^{-\frac{1}{2}}\|_1 \le \frac{n}{c}$. Hence, we have the ergodic sublinear convergence $O(\frac{1}{\sqrt{K-k_0+1}})$; (ii) The existence of $c > 0$ such that $\widetilde{v}_i^{(k_0-1)} \ge c^2$, $\forall i \in [n]$ almost surely is a mild assumption. If k_0 is large, then it is likely that $\widetilde{v}_i^{(k_0-1)} \ge \widehat{v}_i^{(k_0-1)} \gtrapprox (1-\beta_2) G_{i,\infty}^2$, where $G_{i,\infty}$ is an almost-sure bound on $|\nabla_i f(\mathbf{x};\xi)|$; (iii) Suppose $K \ge 2$ and $k_0 = \lceil \frac{K}{2} \rceil$. If $\alpha = O(1)$ and $\tau = o(K^{\frac{1}{4}})$, then $\frac{C_2}{c}(\sqrt{C_1} + \frac{C_2}{c}) \ll C_1$ when K is large. Hence, we observe from (4.20) that in this case, the delay will just slightly affect the convergence speed, and we can achieve nearly-linear speed-up.

5 Numerical experiments

In this section, we conduct numerical experiments on the proposed algorithm APAM under both shared-memory and distributed settings. We compare APAM to the non-parallel and sync-parallel versions of AMSGrad, and also to the async-parallel nonadaptive SGM. Notice that AMSGrad has been shown in the literature to converge significantly faster than a non-adaptive SGM or a momentum SGM, and in addition, it performs similarly well as Adam, another popularly used adaptive SGM not guaranteed to converge. Both async- and sync-parallel methods are implemented in C++, using openMP for shared-memory parallelization and using MPI for distributed communication. They are also implemented in Python for tests with large-scale datasets, using MPI4PY for distributed communication. Tests in Sects. 5.1–5.3 are run with the C++ implementation on a Dell workstation with 32 CPU cores, 64 GB memory, and two Quadro RTX 5000 GPUs. Tests in Sects. 5.4–5.5 are run with the Python implementation on the same workstation.

Table 2 Characteristics of the tested datasets

Name	Train samples	Test samples	Features	Classes
rcv1	20,242	677,399	47,236	2
MNIST	60,000	10,000	28×28	10
Cifar10	50,000	10,000	$32 \times 32 \times 3$	10
CINIC10	180,000	90,000	$32 \times 32 \times 3$	10
Imagenet32×32	1,281,167	50,000	$32 \times 32 \times 3$	1000

In our tests, we use five datasets: rcv1 from LIBSVM [5], MNIST [22], Cifar10 [20], CINIC10 [10], and Imagenet32 $\times$ 32 [9]. Their characteristics are listed in Table 2.

5.1 Performance of APAM

In this subsection, we demonstrate the convergence behavior and parallelization speed-up of APAM on solving both convex and non-convex problems. We apply APAM to solve the logistic regression (LR) problem and to train a 2-layer fully-connected neural network (NN). The LR problem is convex while the neural network training is non-convex. For the LR problem, we use the rcv1 data, and for the 2-layer NN, we use the MNIST data. We set the number of neurons in the hidden layer to 50 in the 2-layer NN, and we use the hyperbolic tangent function as the activation. The initial iterates for both problems are set as the standard Gaussian. While computing a stochastic gradient, the mini-batch size is set to 64 and 32 respectively for the two problems. We set the learning rate to $\alpha_k = 10^{-2}, \forall\, k$ for the LR problem and $\alpha_k = 5 \times 10^{-4}, \forall\, k$ for the 2-layer NN training. The weight parameters are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in this test and also all the other tests.

For both problems, we report the wall-clock time and prediction accuracy on the testing data. In addition, we report the objective error, i.e., the distance of the objective value to the optimal value, for the convex LR problem and the training accuracy for the non-convex 2-layer NN problem. The results for the LR problem are shown in Fig. 3 and those for the 2-layer NN training in Fig. 4. From the results, we see that the convergence speed of APAM, measured by the objective error or prediction accuracy versus epoch number, keeps almost the same when the number of threads used for the shared-memory parallel computing changes. This observation indicates that the convergence speed of APAM is just slightly affected by the information delay. For both problems, over 16x speed-up is achieved in terms of the running time while 32 threads are used.

5.2 Comparison to the nonadaptive SGM

In this subsection, we compare APAM to the async-parallel nonadaptive SGM. The latter method is implemented in a way similar to how we implement APAM but with a nonadaptive update. We test the two methods on training the 2-layer neural network

in the previous subsection and the LeNet5 network [22] by using the MNIST dataset. LeNet5 has 2 convolutional, 2 max-pooling, and 3 fully-connected layers. For the 2-layer network, we use openMP shared-memory parallelism on the two methods. The mini-batch size in computing a stochastic gradient is set to 32 for both methods. The parameters of APAM are set the same as those in the previous subsection, and the learning rate of the nonadaptive SGM is tuned to 10^{-3} for the highest testing accuracy. For the LeNet5 network, we conduct distributed computing with MPI. The mini-batch size is set to 40 for both methods, and the learning rate is tuned to 10^{-4} and 10^{-3} respectively for APAM and the async-parallel nonadaptive SGM, for the highest testing accuracy.

For the openMP implementation, we compare the performance of the two methods by running them with 1, 8, or 32 threads. For the MPI implementation, we compare their performance with one master process and 1, 5, or 20 worker processes. When one thread or one worker is used, the methods become nonparallel. Both methods are run to 200 epochs. The results are shown in Fig. 5 for the openMP implementation and in Fig. 6 for the MPI implementation. Because the nonadaptive SGM update is cheaper than the APAM update, we plot the accuracy versus the running time. From the convergence curves, we see that the convergence speed of both APAM and the async-parallel nonadaptive SGM is slightly affected by the information delay. In addition, we see that APAM gives significantly higher training and testing accuracy than the nonadaptive SGM within the same amount of running time.

5.3 Comparison to sync-parallel method

In this subsection, we compare APAM to its sync-parallel counterpart. We test them on training the 2-layer neural network used in the previous two subsections and on training the AllCNN network in [38] without data augmentation. The MNIST data is used for the 2-layer network and Cifar10 for the AllCNN network. AllCNN has 9 convolutional

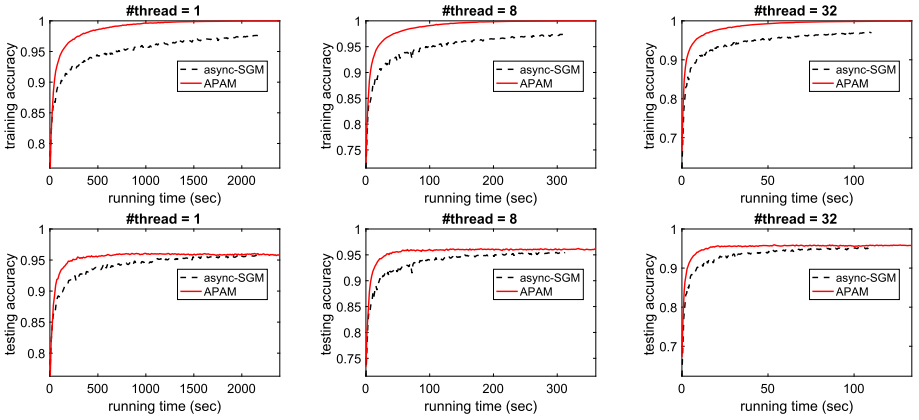


Fig. 5 Training and testing accuracy by APAM and the async-parallel nonadaptive SGM with openMP for a 2-layer fully-connected neural network on MNIST

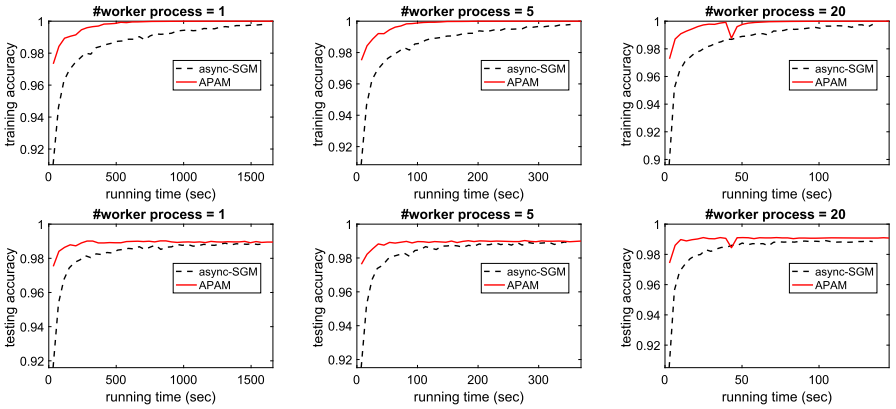


Fig. 6 Training and testing accuracy by APAM and the async-parallel nonadaptive SGM with MPI for the LeNet5 neural network on MNIST dataset

layers. The parameters of APAM and its sync-parallel counterpart are set to the same values. On training the 2-layer network, we adopt the same parameter settings as those in the previous two subsection. On the AllCNN, we conduct distributed computing with MPI. We set the mini-batch size to 40 and tune the learning rate to $\alpha_k = 10^{-4}, \forall\, k$ based on testing accuracy.

The running time for the 2-layer network is shown in Fig. 7and the results for the AllCNN in Fig. 8. Figure 4 shows that on the 2-layer network, APAM gives almost the same accuracy curves while the number of threads used in the training changes. Hence, the results in Figs. 4 and 7 indicate that APAM can achieve significantly higher parallelization speed-up than its sync-parallel counterpart to reach the same training/testing accuracy, especially when 16 or 32 threads are used. The sync-parallel method achieves lower paralleization speed-up by using 32 threads than that by using 8 or 16 threads. This is possibly because of memory congestion. For the AllCNN, we see that in the beginning, APAM produces lower accuracy as more worker processes are used, and this should be because the delay slows down the convergence speed. However, to reach the final highest accuracy, APAM with different number of worker processes takes almost the same number of epochs. Therefore, the results indicate that APAM again has significantly higher speed-up than its sync-parallel counterpart to reach the highest training/testing accuracy, especially when 10 or 20 worker processes are used.

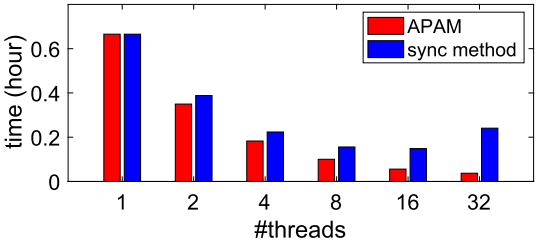


Fig. 7 Running time (h) of APAM and the sync-parallel AMSGrad with openMP by different number of threads for training a 2-layer fully-connected network on MNIST dataset. The training/testing accuracy results are shown in Fig. 4

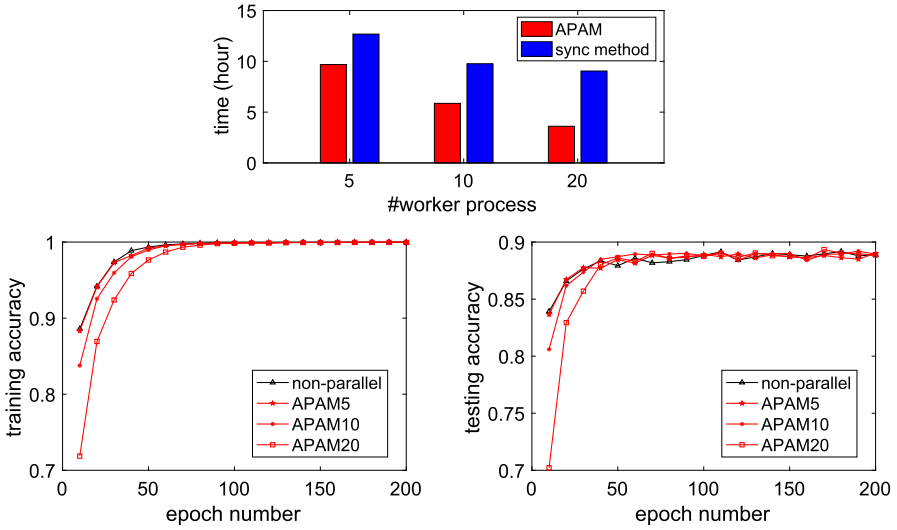


Fig. 8 Running time (h) and prediction accuracy by APAM and the sync-parallel AMSGrad with MPI implementation for training the AllCNN network without data augmentation on Cifar10 dataset

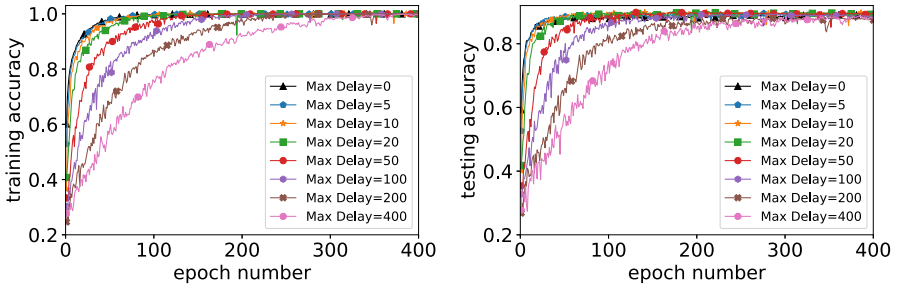


Fig. 9 Predication accuracy by APAM for training the AllCNN network on Cifar10 dataset with Python implementation and artificial delay

5.4 Results with artificial delay

In this subsection, we test the effect of the delay in our algorithm APAM, by artificially injecting different delays like [2]. For a given maximum delay τ, we artificially select the delay τ_k in iteration k from $\{0, 1, ..., \min\{\tau, k\}\}$ uniformly at random, i.e., the stochastic gradient $\mathbf{g}^{(k)}$ is evaluated at an iterate that is selected from $\{\mathbf{x}^{(k)}, \mathbf{x}^{(k-1)}, \ldots, \mathbf{x}^{(k-\min\{\tau,k\})}\}$ uniformly at random. We test APAM on training the AllCNN network on Cifar10 with different maximum delays and the same parameter settings as those in the previous subsection.

Figure 9 plots the curves for different values of τ. In all cases, APAM can converge to almost the same final highest accuracy. When $\tau \leq 20$, APAM takes almost the same number of epochs to reach the final highest accuracy as the no-delay case. When $\tau \geq 50$, the negative effect of delay on the convergence becomes obvious. APAM

with a larger maximum delay converges slower and needs more epochs to achieve the final highest accuracy. APAM with the maximum delay of 200 converges the slowest but it can still achieve the final highest accuracy after about 300 epochs.

5.5 Tests on larger datasets

In this subsection, we test APAM and its sync-parallel counterpart on larger neural networks and larger datasets. We train two networks: Resnet18 [17] that is a deep residual network with 18 convolutional layers, and WRN-28-5 [45] that is a wide residual network with 28 convolutional layers and whose widening factor is 5. Resnet18 is used for classifying the CINIC10 data, with the mini-batch size set to 80 and the learning rate tuned to 10^{-4}. WRN-28-5 is used for classifying the Imagenet32 $\times$ 32 data, with the mini-batch size set to 100 and the learning rate tuned to 10^{-3}.

The training is first run on CPUs to compare the time of APAM and its sync counterpart. Because of the problem size, it takes very long time for one update, and we only run the training to one epoch. Figure 10 shows the running time. From the figure, we see again that APAM has significantly higher speed-up than its sync-parallel counterpart. In more details, the running time for both trainings by APAM decreases as the number of workers increases, and it is reduced almost by a half as the workers increase from 5 to 10 and from 10 to 20. However, for the sync-parallel counterpart, the speed-up is only observed when the number of workers increases from 1 to 5, and as the number of workers further increases, it takes longer time for training Resnet18 on the CINIC10 data.

Then we run APAM with two GPUs to see how the delay affects the convergence. As there are only two GPUs, we assign $\lceil \frac{p}{2} \rceil$ workers on one GPU and $\lfloor \frac{p}{2} \rfloor$ workers on the other one, if p workers are employed. Due to the memory limitation, p is set up to 10. Figures 11 and 12 show the prediction accuracy of the training on Resnet18 and WRN-28-5 respectively. From the figures, we see that APAM produces lower accuracy for the beginning epochs as more workers are used. This indicates that the delay slows down the convergence speed. Nevertheless, APAM achieves almost the same final highest accuracy with different numbers of workers. It is worth to mention that to train the models on the large datasets for many epochs takes a long time, even on GPUs. The number of epochs is selected such that the training takes about one day

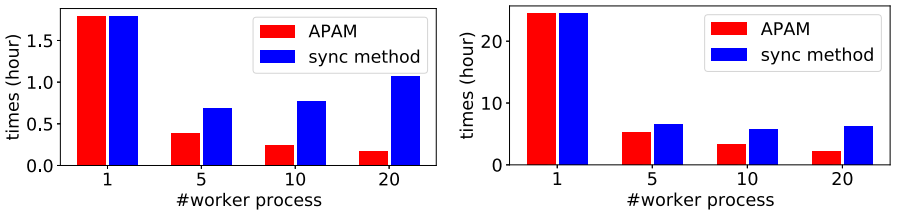


Fig. 10 Running time (h) on CPU by APAM and the sync-parallel AMSGrad with Python and MPI4PY implementation, for training the Resnet18 network on CINIC10 dataset (Left) and WRN-28-5 on Imagenet32 $\times$ 32 dataset (Right); both for one epoch

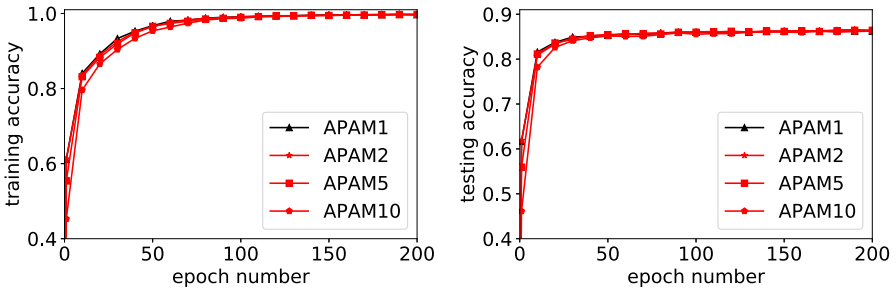


Fig. 11 Prediction accuracy by APAM and the sync-parallel AMSGrad with Python and MPI4PY implementation for training the Resnet18 network on CINIC10 dataset

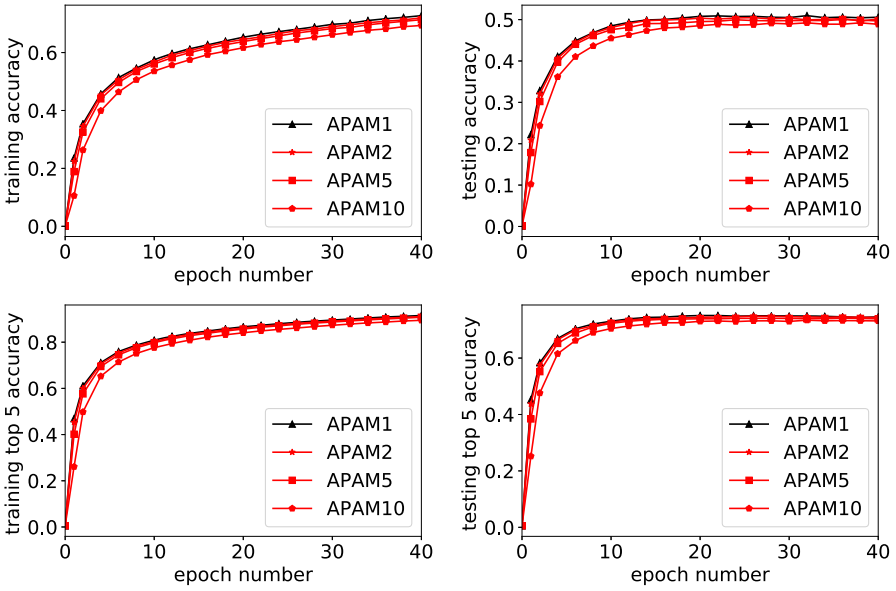


Fig. 12 Prediction accuracy by APAM and the sync-parallel AMSGrad with Python and MPI4PY implementation for training the WRN-28-5 network on Imagenet32 $\times$ 32 dataset. The first row is about the conventional accuracy that measures the proportion of images for which the predicted class (the one with the highest probability) matches the true class. The second row is about the top 5 accuracy that measures the proportion of images for which one of the five classes with top 5 highest probability matches the true class

by using both GPUs. This way, we could run to 200 epochs for training Resnet18 on CINIC10 and only 40 epochs for training WRN-28-5 on Imagenet32 $\times$ 32.

6 Concluding remarks

We have presented an asynchronous parallel adaptive stochastic gradient method, named APAM, based on AMSGrad. Convergence rate results are established for both constrained convex and unconstrained non-convex cases. The results show that the

delay has little effect on the convergence speed, if it is upper bounded by $\tau = o(K^{\frac{1}{4}})$, where K is the maximum number of iterations. Numerical experiments on both convex and non-convex machine learning problems demonstrate significant advantages of the proposed method over its synchronous counterpart and also an asynchronous parallel nonadaptive method, in both shared-memory and distributed environment.

Acknowledgements The authors would like to thank three anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper and also for the careful testing on our codes.

Funding The work of Y. Xu, Y. Yan and C. Sutcher-Shepard is partly supported by NSF grant DMS-2053493 and the Rensselaer-IBM AI Research Collaboration, part of the IBM AI Horizons Network. J. Chen is supported in part by DOE Award DE-OE0000910.

Data availability statement All data analyzed during this study are publicly available. URLs are included in the referred papers.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Code availability The full code was made available for review and has been released at https://github.com/RPI-OPT/APAM.

Proofs of lemmas in Sect. 3

Proof of Lemma 1 From the update of $\mathbf{x}$ in (1.5), we have the optimality condition

$$\mathbf{0} \in \mathcal{N}_X(\mathbf{x}^{(k+1)}) + \sqrt{\widehat{\mathbf{v}}^{(k)}}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) + \alpha_k \mathbf{m}^{(k)},$$

where $\mathcal{N}_X(\mathbf{x})$ denotes the normal cone of X at $\mathbf{x}$. Hence, it follows

$$\left\langle \mathbf{x}^{(k+1)} - \mathbf{x}, \sqrt{\widehat{\mathbf{v}}^{(k)}}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) + \alpha_k \mathbf{m}^{(k)} \right\rangle \leq 0, \ \forall \mathbf{x} \in X. \tag{7.1}$$

By the update of $\mathbf{m}$ in (1.2), it holds

$$\begin{aligned} &\left\langle \mathbf{x}^{(k+1)} - \mathbf{x}, \mathbf{m}^{(k)} \right\rangle \\ &= \left\langle \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}, \mathbf{m}^{(k)} \right\rangle + \left\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{m}^{(k)} \right\rangle \\ &= \left\langle \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}, \mathbf{m}^{(k)} \right\rangle + (1-\beta_1) \left\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{g}^{(k)} \right\rangle + \beta_1 \left\langle \mathbf{x}^{(k)} - \mathbf{x}, \mathbf{m}^{(k-1)} \right\rangle. \end{aligned}$$

Recursively using the above relation, we have

$$\left\langle \mathbf{x}^{(k+1)} - \mathbf{x}, \mathbf{m}^{(k)} \right\rangle = \sum_{j=1}^{k} \beta_1^{k-j} \left(\langle \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \mathbf{m}^{(j)} \rangle + (1-\beta_1) \langle \mathbf{x}^{(j)} - \mathbf{x}, \mathbf{g}^{(j)} \rangle \right). \tag{7.2}$$

In addition, it holds

$$\begin{aligned} &\left\langle \mathbf{x}^{(k+1)} - \mathbf{x}, \sqrt{\widehat{\mathbf{v}}^{(k)}} (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) \right\rangle \\ &= \frac{1}{2} \left(\|\mathbf{x}^{(k+1)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} - \|\mathbf{x}^{(k)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} + \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} \right). \end{aligned}$$

Substituting the above two equations into (7.1) gives

$$\begin{aligned} &\alpha_k \sum_{j=1}^{k} \beta_1^{k-j} \left(\langle \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \mathbf{m}^{(j)} \rangle + (1-\beta_1) \langle \mathbf{x}^{(j)} - \mathbf{x}, \mathbf{g}^{(j)} \rangle \right) \\ &\leq -\frac{1}{2} \left(\|\mathbf{x}^{(k+1)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} - \|\mathbf{x}^{(k)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} + \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} \right). \end{aligned} \tag{7.3}$$

By the Young's inequality, we have

$$\begin{aligned} &\sum_{k=1}^{t} \alpha_k \sum_{j=1}^{k} \beta_1^{k-j} \langle \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \mathbf{m}^{(j)} \rangle \\ &= \sum_{j=1}^{t} \sum_{k=j}^{t} \alpha_k \beta_1^{k-j} \langle \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \mathbf{m}^{(j)} \rangle \\ &\geq \sum_{j=1}^{t} \left(\sum_{k=j}^{t} \alpha_k \beta_1^{k-j} \right) \left(- \frac{\|\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}\|^2_{(\widehat{\mathbf{v}}^{(j)})^{\frac{1}{2}}}}{2 \sum_{k=j}^{t} \alpha_k \beta_1^{k-j}} - \frac{\sum_{k=j}^{t} \alpha_k \beta_1^{k-j}}{2} \|\mathbf{m}^{(j)}\|^2_{(\widehat{\mathbf{v}}^{(j)})^{-\frac{1}{2}}} \right). \end{aligned}$$

Since $\sum_{k=j}^{t} \alpha_k \beta_1^{k-j} \leq \frac{\alpha_j}{1-\beta_1}$, the above inequality implies

$$\begin{aligned} &\sum_{k=1}^{t} \alpha_k \sum_{j=1}^{k} \beta_1^{k-j} \langle \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \mathbf{m}^{(j)} \rangle \\ &\geq -\sum_{j=1}^{t} \left(\frac{\|\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}\|^2_{(\widehat{\mathbf{v}}^{(j)})^{\frac{1}{2}}}}{2} + \frac{\alpha_j^2}{2(1-\beta_1)^2} \|\mathbf{m}^{(j)}\|^2_{(\widehat{\mathbf{v}}^{(j)})^{-\frac{1}{2}}} \right). \end{aligned} \tag{7.4}$$

In addition, noting $\widehat{\mathbf{v}}^{(k)} \geq \widehat{\mathbf{v}}^{(k-1)}$ for all k, we have

$$-\sum_{k=1}^{t} \left(\|\mathbf{x}^{(k+1)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} - \|\mathbf{x}^{(k)} - \mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}} \right)$$

$$= -\|\mathbf{x}^{(t+1)}-\mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(t)}}}+\sum_{k=2}^{t}\|\mathbf{x}^{(k)}-\mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(k)}}-\sqrt{\widehat{\mathbf{v}}^{(k-1)}}}+\|\mathbf{x}^{(1)}-\mathbf{x}\|^2_{\sqrt{\widehat{\mathbf{v}}^{(1)}}}$$

$$\le D_\infty^2\left(\sum_{k=2}^{t}\|\sqrt{\widehat{\mathbf{v}}^{(k)}}-\sqrt{\widehat{\mathbf{v}}^{(k-1)}}\|_1+\|\sqrt{\widehat{\mathbf{v}}^{(1)}}\|_1\right)=D_\infty^2\|\sqrt{\widehat{\mathbf{v}}^{(t)}}\|_1. \tag{7.5}$$

Now summing (7.3) over $k=1$ to t, and using (7.4) and (7.5), we obtain the desired result. □

Proof of Lemma 2 For each $i\in[n]$, let $G_i^{(k)}=\max_{j\le k}|g_i^{(j)}|$, and $\mathbf{G}^{(k)}$ be the vector with the i-th component $G_i^{(k)}$. Note that for each $k\ge 1$ and each $i\in[n]$, we have $\widehat{v}_i^{(k)}=\max\{\widehat{v}_i^{(k-1)},\ v_i^{(k)}\}=\max_{j\le k}v_i^{(j)}$, and in addition, $v_i^{(j)}=\sum_{j=1}^{j}(1-\beta_2)\beta_2^{j-J}(g_i^{(J)})^2$. Hence,

$$\widehat{v}_i^{(k)}=\max_{j\le k}\sum_{J=1}^{j}(1-\beta_2)\beta_2^{j-J}(g_i^{(J)})^2, \tag{7.6}$$

and thus $\widehat{v}_i^{(k)}\ge(1-\beta_2)(G_i^{(k)})^2$. Therefore, noticing

$$\mathbf{m}^{(k)}=\sum_{j=1}^{k}(1-\beta_1)\beta_1^{k-j}\mathbf{g}^{(j)}, \tag{7.7}$$

we have

$$\|\mathbf{m}^{(k)}\|_{(\widehat{\mathbf{v}}^{(k)})^{-\frac{1}{2}}}=\|\mathbf{m}^{(k)}\oslash(\widehat{\mathbf{v}}^{(k)})^{\frac{1}{4}}\|\le\frac{1}{(1-\beta_2)^{\frac{1}{4}}}\|\mathbf{m}^{(k)}\oslash\sqrt{\mathbf{G}^{(k)}}\|$$

$$\le\frac{1}{(1-\beta_2)^{\frac{1}{4}}}\sum_{j=1}^{k}(1-\beta_1)\beta_1^{k-j}\left\|\mathbf{g}^{(j)}\oslash\sqrt{\mathbf{G}^{(k)}}\right\|,$$

and thus by the Cauchy–Schwarz inequality, it holds

$$\|\mathbf{m}^{(k)}\|^2_{(\widehat{\mathbf{v}}^{(k)})^{-\frac{1}{2}}}\le\frac{(1-\beta_1)^2}{(1-\beta_2)^{\frac{1}{2}}}\left(\sum_{j=1}^{k}\beta_1^{k-j}\right)\left(\sum_{j=1}^{k}\beta_1^{k-j}\left\|\mathbf{g}^{(j)}\oslash\sqrt{\mathbf{G}^{(k)}}\right\|^2\right)$$

$$\le\frac{1-\beta_1}{(1-\beta_2)^{\frac{1}{2}}}\sum_{j=1}^{k}\beta_1^{k-j}\left\|\mathbf{g}^{(j)}\oslash\sqrt{\mathbf{G}^{(k)}}\right\|^2.$$

Now note that

$$\left\|\mathbf{g}^{(j)}\oslash\sqrt{\mathbf{G}^{(k)}}\right\|^2=\sum_{i=1}^{n}\frac{|g_i^{(j)}|^2}{G_i^{(k)}}\le\sum_{i=1}^{n}|g_i^{(j)}|=\|\mathbf{g}^{(j)}\|_1.$$

Together from the above two inequalities and Assumption 2, it follows that

$$\mathbb{E}\|\mathbf{m}^{(k)}\|^2_{(\widehat{\mathbf{v}}^{(k)})^{-\frac{1}{2}}} \le \frac{1-\beta_1}{(1-\beta_2)^{\frac{1}{2}}}\sum_{j=1}^{k}\beta_1^{k-j}\mathbb{E}\|\mathbf{g}^{(j)}\|_1 \le \frac{1-\beta_1}{(1-\beta_2)^{\frac{1}{2}}}\sum_{j=1}^{k}\beta_1^{k-j}G_1,$$

which implies the result in (3.2). □

Proof of Lemma 3 Since X is separable and $x_i^{(k+1)} = x_i^{(k)}$ if $\widehat{v}_i^{(k)} = 0$, we have $\mathbf{x}^{(k+1)} = \mathrm{Proj}_X\left(\mathbf{x}^{(k)} - \alpha_k\mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\right)$. In addition, notice that $\mathbf{x}^{(k)} = \mathrm{Proj}_X\left(\mathbf{x}^{(k)}\right)$, and thus the desired result follows from the non-expansiveness of the projection onto a convex set. □

Proof of Lemma 4 For each $i \in [n]$, let $G_i^{(k)} = \max_{j\le k}|g_i^{(j)}|$, and $\mathbf{G}^{(k)}$ be the vector with the i-th component $G_i^{(k)}$. Then it follows from (7.6) that $\widehat{v}_i^{(k)} \ge (1-\beta_2)(G_i^{(k)})^2$. Hence, for $j \le k$,

$$\|\mathbf{g}^{(j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\|^2 \le \frac{1}{1-\beta_2}\sum_{i=1}^{n}\frac{(g_i^{(j)})^2}{(G_i^{(k)})^2} \le \frac{\|\mathbf{g}^{(j)}\|_0}{1-\beta_2},$$

which gives (3.11a). Furthermore, by (7.7), it holds

$$\left\|\mathbf{m}^{(k)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\right\| \le \sum_{j=1}^{k}(1-\beta_1)\beta_1^{k-j}\left\|\mathbf{g}^{(j)} \oslash \sqrt{\widehat{\mathbf{v}}^{(k)}}\right\| \le \sum_{j=1}^{k}(1-\beta_1)\beta_1^{k-j}\frac{\sqrt{\|\mathbf{g}^{(j)}\|_0}}{\sqrt{1-\beta_2}},$$

which proves (3.11b). The above inequality together with the Cauchy–Schwarz inequality implies (3.11c). Hence, we complete the proof. □

Proofs of lemmas in Sect. 4

Proof of Lemma 5 From (7.7), we have $m_i^{(k)} = (1-\beta_1)\sum_{j=1}^{k}\beta_1^{k-j}g_i^{(j)}$, and thus applying triangle inequality and using the definition of $\mathbf{\Gamma}$ in (4.2) lead to $|m_i^{(k)}| \le (1-\beta_1^k)\Gamma_i \le \Gamma_i$. A similar argument gives $\widehat{v}_i^{(k)} \le \Gamma_i^2$. When Assumption 3 holds, we know that $\|\mathbf{g}^{(k)}\|_\infty \le G_\infty$ almost surely, and that $\|\nabla F(\mathbf{x}^{(k)})\|_\infty \le G_\infty$, for all $k \in [K]$, which leads to the second part of this lemma. □

Proof of Lemma 6 From (4.3), we have that for $k \ge 1$,

$$\begin{aligned}\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)} &= \frac{1}{1-\beta_1}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) - \frac{\beta_1}{1-\beta_1}(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})\\ &= -\frac{1}{1-\beta_1}\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{m}^{(k)} + \frac{\beta_1}{1-\beta_1}\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{m}^{(k-1)},\end{aligned}$$

where in the second equation, we have used (4.1) and Remark 4. With $k = 1$, the above equation gives (4.4) by utilizing the update of $\mathbf{m}^{(1)}$. Furthermore, it gives, by plugging the update of $\mathbf{m}^{(k)}$,

$$\begin{aligned}
&\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)} \\
&= \frac{-1}{1-\beta_1}\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}(\beta_1\mathbf{m}^{(k-1)} + (1-\beta_1)\mathbf{g}^{(k)}) + \frac{\beta_1}{1-\beta_1}\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{m}^{(k-1)} \\
&= \frac{\beta_1}{1-\beta_1}\left[\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\right]\mathbf{m}^{(k-1)} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)} \\
&= \frac{\beta_1}{1-\beta_1}\left[\mathbf{I} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\alpha_{k-1}^{-1}(\widetilde{\mathbf{V}}^{(k-1)})^{\frac{1}{2}}\right]\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{m}^{(k-1)} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}.
\end{aligned}$$

The second equation of the above is exactly (4.5), and the last equation gives (4.6). □

Proof of Lemma 7 Inner producting $\nabla F(\mathbf{x}^{(k)})$ with both sides of (4.5) gives

$$\begin{aligned}
\nabla F(\mathbf{x}^{(k)})^\top(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) = \tfrac{\beta_1}{1-\beta_1}\nabla F(\mathbf{x}^{(k)})^\top\left[\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\right]\mathbf{m}^{(k-1)} \\
- \nabla F(\mathbf{x}^{(k)})^\top\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}. \qquad (8.1)
\end{aligned}$$

We bound the first term on the right-hand-side of (8.1) by Definition 2 and Lemma 5 as follows:

$$\begin{aligned}
&\nabla F(\mathbf{x}^{(k)})^\top\left[\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\right]\mathbf{m}^{(k-1)} \\
&= \sum_{i=1}^n \nabla_i F(\mathbf{x}^{(k)})\left[\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right]m_i^{(k-1)} \\
&\le \sum_{i=1}^n \Phi_i\left|\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right|\Gamma_i \\
&= \sum_{i=1}^n \Gamma_i\Phi_i\left[\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right], \qquad (8.2)
\end{aligned}$$

where the last equation follows because $\alpha_{k-1}(\widetilde{\mathbf{v}}^{(k-1)})^{-\frac{1}{2}} \ge \alpha_k(\widetilde{\mathbf{v}}^{(k)})^{-\frac{1}{2}} > \mathbf{0}$ component-wisely. Similarly, we can bound the second term on the right-hand-side of (8.1) as follows:

$$\begin{aligned}
&-\nabla F(\mathbf{x}^{(k)})^\top\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)} \\
&= -\nabla F(\mathbf{x}^{(k)})^\top\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{g}^{(k)} \\
&\quad + \nabla F(\mathbf{x}^{(k)})^\top\left[\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\right]\mathbf{g}^{(k)} \\
&= -\nabla F(\mathbf{x}^{(k)})^\top\alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{g}^{(k)} \\
&\quad + \sum_{i=1}^n \nabla_i F(\mathbf{x}^{(k)})\left[\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right]g_i^{(k)}
\end{aligned}$$

$$\leq -\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{g}^{(k)} + \sum_{i=1}^{n} \Phi_i \left|\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right| \Gamma_i$$

$$= -\nabla F(\mathbf{x}^{(k)})^\top \alpha_{k-1}(\widetilde{\mathbf{V}}^{(k-1)})^{-\frac{1}{2}}\mathbf{g}^{(k)} + \sum_{i=1}^{n} \Gamma_i \Phi_i \left[\alpha_{k-1}(\widetilde{v}_i^{(k-1)})^{-\frac{1}{2}} - \alpha_k(\widetilde{v}_i^{(k)})^{-\frac{1}{2}}\right]. \tag{8.3}$$

Now substituting (8.2) and (8.3) into (8.1) yields (4.7). □

Proof of Lemma 10 From (4.8) and the fact $(a+b)^2 \leq 4a^2 + \frac{4}{3}b^2$, $\forall\, a, b \in \mathbb{R}$, the inequality in (4.11) immediately follows. By the Cauchy–Schwarz inequality, and also (4.9) and(4.8), it holds

$$\begin{aligned}
&\left(\nabla F(\mathbf{z}^{(k)}) - \nabla F(\mathbf{x}^{(k)})\right)^\top (\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}) \\
&\leq \|\nabla F(\mathbf{z}^{(k)}) - \nabla F(\mathbf{x}^{(k)})\| \cdot \|\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\| \\
&\leq \frac{L\beta_1}{1-\beta_1}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\| \left(\frac{\beta_1}{1-\beta_1}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\| + \|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|\right) \\
&= \frac{L\beta_1^2}{(1-\beta_1)^2}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\|^2 + \frac{\beta_1 L}{1-\beta_1}\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\| \cdot \|\alpha_k(\widetilde{\mathbf{V}}^{(k)})^{-\frac{1}{2}}\mathbf{g}^{(k)}\|.
\end{aligned}$$

Now using the Young's inequality, we have (4.10) from the above inequality. □

References

1. Agarwal, A., Duchi, J.C.: Distributed delayed stochastic optimization. In: Advances in Neural Information Processing Systems, pp. 873–881 (2011)
2. Assran, M.S., Rabbat, M.G.: Asynchronous gradient push. IEEE Trans. Autom. Control **66**(1), 168–183 (2020)
3. Bäckström, K., Papatriantafilou, M., Tsigas, P.: Mindthestep-asyncPSGD: adaptive asynchronous parallel stochastic gradient descent. In: 2019 IEEE International Conference on Big Data (Big Data), pp 16–25. IEEE (2019)
4. Bertsekas, D.P., Tsitsiklis, J.N.: Some aspects of parallel and distributed iterative algorithms—a survey. Automatica **27**(1), 3–21 (1991)
5. Chang, C.-C., Lin, C.-J.: Libsvm: a library for support vector machines. ACM Trans. Intell. Syst. Technol. (TIST) **2**(3), 1–27 (2011)
6. Chen, J. Gu, Q.: Closing the generalization gap of adaptive gradient methods in training deep neural networks. arXiv preprint arXiv:1806.06763v1 (2018)
7. Chen, J., Zhou, D., Tang, Y., Yang, Z., Cao, Y., Gu, Q.: Closing the generalization gap of adaptive gradient methods in training deep neural networks. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp. 3267–3275 (2021)
8. Chen, X., Liu, S., Sun, R., Hong, M.: On the convergence of a class of Adam-type algorithms for non-convex optimization. In: International Conference on Learning Representations (2019)
9. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A down sampled variant of imagenet as an alternative to the CIFAR datasets. arXiv preprint arXiv:1707.08819 (2017)
10. Darlow, L.N., Crowley, E.J., Antoniou, A., Storkey, A.J.: Cinic-10 is not imagenet or cifar-10. arXiv preprint arXiv:1810.03505 (2018)
11. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q., Ng, A.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, pp. 1223–1231 (2012)

12. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
13. Fang, B., Klabjan, D.: Convergence analyses of online ADAM algorithm in convex setting and two-layer ReLU neural network. arXiv preprint arXiv:1905.09356 (2019)
14. Feyzmahdavian, H.R., Aytekin, A., Johansson, M.: An asynchronous mini-batch algorithm for regularized stochastic optimization. IEEE Trans. Autom. Control **61**(12), 3740–3754 (2016)
15. Ghadimi, S., Lan, G.: Stochastic first-and zeroth-order methods for nonconvex stochastic programming. SIAM J. Optim. **23**(4), 2341–2368 (2013)
16. Guan, N., Shan, L., Yang, C., Xu, W., Zhang, M.: Delay compensated asynchronous ADAM algorithm for deep neural networks. In: 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), pp. 852–859. IEEE (2017)
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
18. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016)
19. Kingma, D.P., Ba, J.: ADAM: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images Computer Science Department, University of Toronto, Tech. Rep. (2009)
21. Leblond, R., Pedregosa, F., Lacoste-Julien, S.: Improved asynchronous parallel optimization analysis for stochastic incremental methods. J. Mach. Learn. Res. **19**(1), 3140–3207 (2018)
22. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
23. Lian, X., Huang, Y., Li, Y., Liu, J.: Asynchronous parallel stochastic gradient for nonconvex optimization. In: Advances in Neural Information Processing Systems pp. 2737–2745 (2015)
24. Lian, X., Zhang, W., Zhang, C., Liu, J.: Asynchronous decentralized parallel stochastic gradient descent. In: International Conference on Machine Learning, pp. 3043–3052 (2018)
25. Liu, J., Wright, S., Ré, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. In: International Conference on Machine Learning, pp. 469–477 (2014)
26. Liu, J., Zhang, C., et al.: Distributed learning systems with first-order methods. Found Trends® Databases **9**(1), 1–100 (2020)
27. Luo, L., Xiong, Y., Liu, Y.: Adaptive gradient methods with dynamic bound of learning rate. In: International Conference on Learning Representations (2019)
28. Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., Jordan, M.I.: Perturbed iterate analysis for asynchronous stochastic optimization. SIAM J. Optim. **27**(4), 2202–2229 (2017)
29. Masters, D., Luschi, C.: Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612 (2018)
30. Nazari, P., Tarzanagh, D.A., Michailidis, G.: Dadam: a consensus-based distributed adaptive gradient method for online optimization. IEEE Trans. Signal Process. **70**, 6065–6079 (2022)
31. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM J. Optim. **19**(4), 1574–1609 (2009)
32. Peng, Z., Xu, Y., Yan, M., Yin, W.: ARock: an algorithmic framework for asynchronous parallel coordinate updates. SIAM J. Sci. Comput. **38**(5), A2851–A2879 (2016)
33. Peng, Z., Xu, Y., Yan, M., Yin, W.: On the convergence of asynchronous parallel iteration with unbounded delays. J. Oper. Res. Soc. China **7**(1), 5–42 (2019)
34. Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM J. Control. Optim. **30**(4), 838–855 (1992)
35. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: a lock-free approach to parallelizing stochastic gradient descent. In: Advances in Neural Information Processing Systems, pp. 693–701 (2011)
36. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of ADAM and beyond. In: International Conference on Learning Representations (2018)
37. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**(3), 400–407 (1951)
38. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
39. Sra, S., Yu, A.W., Li, M., Smola, A.: Adadelay: delay adaptive distributed stochastic optimization. In: Artificial Intelligence and Statistics, pp. 957–965 (2016)

40. Tieleman, T., Hinton, G.: RMSProp: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
41. Tran, P.T., Phong, L.T.: On the convergence proof of AMSGrad and a new version. IEEE Access **7**, 61706–61716 (2019)
42. Wang, G., Lu, S., Cheng, Q., Tu, W., Zhang, L.: SAdam: a variant of adam for strongly convex functions. In: International Conference on Learning Representations (2020)
43. Wu, J., Huang, W., Huang, J., Zhang, T.: Error compensated quantized SGD and its applications to large-scale distributed optimization. In: International Conference on Machine Learning, pp. 5325–5333 (2018)
44. Yan, Y., Yang, T., Li, Z., Lin, Q., Yang, Y.: A unified analysis of stochastic momentum methods for deep learning. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, pp. 2955–2961. International Joint Conferences on Artificial Intelligence Organization, 7 (2018)
45. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: British Machine Vision Conference 2016. British Machine Vision Association (2016)
46. Zhou, D., Tang, Y., Yang, Z., Cao, Y., Gu, Q.:. On the convergence of adaptive gradient methods for nonconvex optimization. arXiv preprint arXiv:1808.05671 (2018)