BASGD: Buffered Asynchronous SGD for Byzantine Learning

Yi-Rui Yang [1] **Wu-Jun Li** [1]

Abstract

Distributed learning has become a hot research topic due to its wide application in cluster-based large-scale learning, federated learning, edge computing and so on. Most traditional distributed learning methods typically assume no failure or attack. However, many unexpected cases, such as communication failure and even malicious attack, may happen in real applications. Hence, Byzantine learning (BL), which refers to distributed learning with failure or attack, has recently attracted much attention. Most existing BL methods are synchronous, which are impractical in some applications due to heterogeneous or offline workers. In these cases, asynchronous BL (ABL) is usually preferred. In this paper, we propose a novel method, called buffered asynchronous stochastic gradient descent (BASGD), for ABL. To the best of our knowledge, BASGD is the first ABL method that can resist malicious attack without storing any instances on server. Compared with those methods which need to store instances on server, BASGD has a wider scope of application. BASGD is proved to be convergent, and be able to resist failure or attack. Empirical results show that BASGD significantly outperforms vanilla asynchronous stochastic gradient descent (ASGD) and other ABL baselines when there exists failure or attack on workers.

1. Introduction

Due to the wide application in cluster-based large-scale learning, federated learning (Konevcnỳ et al., 2016; Kairouz et al., 2019), edge computing (Shi et al., 2016) and so on, distributed learning has recently become a hot research topic (Zinkevich et al., 2010; Yang, 2013; Jaggi et al., 2014; Shamir et al., 2014; Zhang & Kwok, 2014; Ma et al., 2015; Lee et al., 2017; Lian et al., 2017; Zhao et al., 2017; Sun et al., 2018; Wangni et al., 2018; Zhao et al., 2018; Zhou et al., 2018; Yu et al., 2019a;b; Haddadpour et al., 2019). Most traditional distributed learning methods are based on stochastic gradient descent (SGD) and its variants (Bottou, 2010; Xiao, 2010; Duchi et al., 2011; Johnson & Zhang, 2013; Shalev-Shwartz & Zhang, 2013; Zhang et al., 2013; Lin et al., 2014; Schmidt et al., 2017; Zheng et al., 2017; Zhao et al., 2018), and typically assume no failure or attack.

However, in real distributed learning applications with multiple networked machines (nodes), different kinds of hardware or software failure may happen. Representative failure includes bit-flipping in the communication media and the memory of some workers (Xie et al., 2019). In this case, small failure on some machines (workers) might cause a distributed learning method to fail. In addition, malicious attack should not be neglected in an open network where the manager (or server) generally has not much control on the workers, such as the cases of edge computing and federated learning. Malicious workers may behave arbitrarily or even adversarially. Hence, *Byzantine learning* (BL), which refers to distributed learning with failure or attack, has attracted much attention (Diakonikolas et al., 2017; Chen et al., 2017; Blanchard et al., 2017; Damaskinos et al., 2018; Baruch et al., 2019; Diakonikolas & Kane, 2019).

Existing BL methods can be divided into two main categories: synchronous BL (SBL) methods and asynchronous BL (ABL) methods. In SBL methods, the learning information, such as the gradient in SGD, of all workers will be aggregated in a synchronous way. On the contrary, in ABL methods the learning information of workers will be aggregated in an asynchronous way. Existing SBL methods mainly take two different ways to achieve resilience against *Byzantine workers* which refer to those workers with failure or attack. One way is to replace the simple averaging aggregation operation with some more robust aggregation operations, such as median and trimmed-mean (Yin et al., 2018). Krum (Blanchard et al., 2017) and ByzantinePGD (Yin et al., 2019) take this way. The other way is to filter the suspicious learning information (gradients) before averaging. Representative examples include ByzantineSGD (Alistarh et al., 2018) and Zeno (Xie et al., 2019). The advantage of SBL methods is that they are relatively simple and easy to be implemented. But SBL methods will result in slow

[1]National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China. Correspondence to: Wu-Jun Li <liwujun@nju.edu.cn>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

convergence when there exist heterogeneous workers. Furthermore, in some applications like federated learning and edge computing, synchronization cannot even be performed most of the time due to the offline workers (clients or edge servers). Hence, ABL is preferred in these cases.

To the best of our knowledge, there exist only two ABL methods: Kardam (Damaskinos et al., 2018) and Zeno++ (Xie et al., 2020). Kardam introduces two filters to drop out suspicious learning information (gradients), which can still achieve good performance when the communication delay is heavy. However, when in face of malicious attack, some work (Xie et al., 2020) finds that Kardam also drops out most correct gradients in order to filter all faulty (failure) gradients. Hence, Kardam cannot resist malicious attack. Zeno++ needs to store some training instances on server for scoring. In some practical applications like federated learning (Kairouz et al., 2019), storing data on server will increase the risk of privacy leakage or even face legal risk. Therefore, under the general setting where server has no access to any training instances, there does not exist ABL methods which can resist malicious attack.

In this paper, we propose a novel method, called buffered asynchronous stochastic gradient descent (BASGD), for ABL. The main contributions are listed as follows:

- To the best of our knowledge, BASGD is the first ABL method that can resist malicious attack without storing any instances on server. Compared with those methods which need to store instances on server, BASGD has a wider scope of application.
- BASGD is theoretically proved to be convergent, and be able to resist failure or attack.
- Empirical results show that BASGD significantly outperforms vanilla asynchronous stochastic gradient descent (ASGD) and other ABL baselines when there exist failure or malicious attack on workers. In particular, BASGD can still converge under malicious attack, when ASGD and other ABL methods fail.

2. Preliminary

In this section, we present the preliminary of this paper, including the distributed learning framework used in this paper and the definition of Byzantine worker.

2.1. Distributed Learning Framework

Many machine learning models, such as logistic regression and deep neural networks, can be formulated as the following finite sum optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{w}; z_i), \tag{1}$$

where $\mathbf{w}$ is the parameter to learn, d is the dimension of parameter, n is the number of training instances, $f(\mathbf{w}; z_i)$ is the empirical loss on the instance z_i. The goal of distributed learning is to solve the problem in (1) by designing learning algorithms based on multiple networked machines.

Although there have appeared many distributed learning frameworks, in this paper we focus on the widely used Parameter Server (PS) framework (Li et al., 2014). In a PS framework, there are several workers and one or more servers. Each worker can only communicate with server(s). There may exist more than one server in a PS framework, but for the problem of this paper servers can be logically conceived as a unity. Without loss of generality, we will assume there is only one server in this paper. Training instances are disjointedly distributed across m workers. Let $\mathcal{D}_k$ denote the index set of training instances on worker_k, we have $\cup_{k=1}^{m}\mathcal{D}_k = \{1, 2, \ldots, n\}$ and $\mathcal{D}_k \cap \mathcal{D}_{k'} = \emptyset$ if $k \neq k'$. In this paper, we assume that server has no access to any training instances. If two instances have the same value, they are still deemed as two distinct instances. Namely, z_i may equal $z_{i'}$ $(i \neq i')$. One popular asynchronous method to solve the problem in (1) under the PS framework is ASGD (Dean et al., 2012) (see Appendix A of supplementary materials for details). In this paper, we assume each worker samples one instance for gradient computation each time. The analysis of mini-batch case is similar.

In PS based ASGD, server is responsible for updating and maintaining the latest parameter. The number of iterations that server has already executed is used as the global logical clock of server. At the beginning, iteration number $t = 0$. Each time a SGD step is executed, t will increase by 1 immediately. The parameter after t iterations is denoted as $\mathbf{w}^t$. If server sends parameters to worker_k at iteration t', some SGD steps may have been excuted before server receives gradient from worker_k next time at iteration t. Thus, we define the *delay* of worker_k at iteration t as $\tau_k^t = t - t'$. Worker_k is *heavily delayed* at iteration t if $\tau_k^t > \tau_{max}$, where τ_{max} is a pre-defined non-negative constant.

2.2. Byzantine Worker

For workers that have sent gradients (one or more) to server at iteration t, we call worker_k *loyal worker* if it has finished all the tasks without any fault and each sent gradient is correctly received by the server. Otherwise, worker_k is called *Byzantine worker*. If worker_k is a Byzantine worker, it means the received gradient from worker_k is not credible, which can be an arbitrary value. In ASGD, there is one received gradient at a time. Formally, we denote the gradient received from worker_k at iteration t as $\mathbf{g}_k^t$. Then, we have:

$$\mathbf{g}_k^t = \begin{cases} \nabla f(\mathbf{w}^{t'}; z_i), & \text{if worker\_}k \text{ is loyal at iteration } t; \\ *, & \text{if worker\_}k \text{ is Byzantine at iteration } t, \end{cases}$$

Algorithm 1 Buffered Asynchronous SGD (BASGD)

Server:
Input: learning rate η, reassignment interval Δ, buffer number B, aggregation function: $Aggr(\cdot)$;
Initialization: initial parameter $\mathbf{w}^0$, learning rate η;
Set buffer: $\mathbf{h}_b \leftarrow \mathbf{0}$, $N_b^t \leftarrow 0$;
Initialize mapping table $\beta_s \leftarrow s$ $(s = 0, 1, \ldots, m-1)$;
Send initial $\mathbf{w}^0$ to all workers;
Set $t \leftarrow 0$, and start the timer;
repeat
  Wait until receiving $\mathbf{g}$ from some worker_s;
  Choose buffer: $b \leftarrow \beta_s \ mod \ B$;
  Let $N_b^t \leftarrow N_b^t + 1$, and $\mathbf{h}_b \leftarrow \frac{(N_b^t - 1)\mathbf{h}_b + \mathbf{g}}{N_b^t}$;
  if $N_b^t > 0$ for each $b \in [B]$ **then**
    Aggregate: $\mathbf{G}^t = Aggr([\mathbf{h}_1, \ldots, \mathbf{h}_B])$;
    Execute SGD step: $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \cdot \mathbf{G}^t$;
    Zero out buffers: $\mathbf{h}_b \leftarrow \mathbf{0}$, $N_b^t \leftarrow 0$ $(b = 1, \ldots, B)$;
    Set $t \leftarrow t + 1$, and restart the timer;
  end if
  if the timer has exceeded Δ seconds **then**
    Zero out buffers: $\mathbf{h}_b \leftarrow \mathbf{0}$, $N_b^t \leftarrow 0$ $(b = 1, \ldots, B)$;
    Modify the mapping table $\{\beta_s\}_{s=0}^{m-1}$ for buffer reassignment, and restart the timer;
  end if
  Send back the latest parameters back to worker_s, no matter whether a SGD step is executed or not.
until stop criterion is satisfied
Notify all workers to stop;

Worker_k: $(k = 0, 1, ..., m-1)$
repeat
  Wait until receiving the latest parameter $\mathbf{w}$ from server;
  Randomly sample an index i from $\mathcal{D}_k$;
  Compute $\nabla f(\mathbf{w}; z_i)$;
  Send $\nabla f(\mathbf{w}; z_i)$ to server;
until receive server's notification to stop

where $0 \leq t' \leq t$, and i is randomly sampled from $\mathcal{D}_k$. '$*$' represents an arbitrary value. Our definition of Byzantine worker is consistent with most previous works (Blanchard et al., 2017; Xie et al., 2019; 2020). Either accidental failure or malicious attack will result in Byzantine workers.

3. Buffered Asynchronous SGD

In synchronous BL, gradients from all workers are received at each iteration. We can compare the gradients with each other, and then filter suspicious ones, or use more robust aggregation rules such as median and trimmed-mean for updating. However, in asynchronous BL, only one gradient is received at a time. Without any training instances stored on server, it is difficult for server to identify whether a received gradient is credible or not.

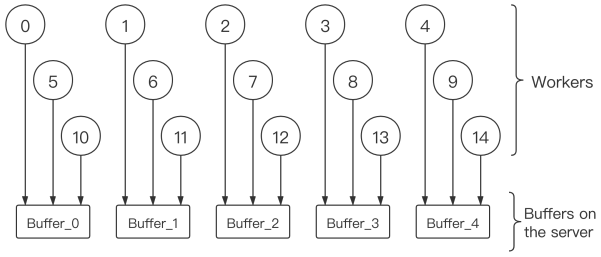


Figure 1. An example of buffers. Circle represents worker, and the number is worker ID. There are 15 workers and 5 buffers. The gradient received from worker_s is stored in buffer_$\{s \bmod 5\}$.

In order to deal with this problem in asynchronous BL, we propose a novel method called buffered asynchronous SGD (BASGD). BASGD introduces B buffers $(0 < B \leq m)$ on server, and the gradient used for updating parameters will be aggregated from these buffers. The detail of the learning procedure of BASGD is presented in Algorithm 1. In this section, we will introduce the three key components of BASGD: buffer, aggregation function, and mapping table.

3.1. Buffer

In BASGD, the m workers do the same job as that in ASGD, while the updating rule on server is modified. More specifically, there are B buffers $(0 < B \leq m)$ on server. When a gradient $\mathbf{g}$ from worker_s is received, it will be temporarily stored in buffer b, where $b = s \ mod \ B$, as illustrated in Figure 1. Only when each buffer has stored at least one gradient, a new SGD step will be executed. Please note that no matter whether a SGD step is executed or not, the server will immediately send the latest parameters back to the worker after receiving a gradient. Hence, BASGD introduces no barrier, and is an asynchronous algorithm.

For each buffer b, more than one gradient may have been received at iteration t. We will store the average of these gradients (denoted by $\mathbf{h}_b$) in buffer b. Assume that there are already $(N-1)$ gradients $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{N-1}$ which should be stored in buffer b, and $\mathbf{h}_{b(old)} = \frac{1}{N-1}\sum_{i=1}^{N-1} \mathbf{g}_i$. When the N-th gradient $\mathbf{g}_N$ is received, the new average value is:

$$\mathbf{h}_{b(new)} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i = \frac{N-1}{N} \cdot \mathbf{h}_{b(old)} + \frac{1}{N} \cdot \mathbf{g}_N.$$

This is the updating rule for each buffer b when a gradient is received. We use N_b^t to denote the total number of gradients stored in buffer b at the t-th iteration. After the parameter $\mathbf{w}$ is updated, all buffers will be zeroed out at once. With the benefit of buffers, server has access to B candidate gradients

when updating parameter. Thus, a more reliable (robust) gradient can be aggregated from the B gradients of buffers, if a proper aggregation function $Aggr(\cdot)$ is chosen.

Please note that from the perspective of workers, BASGD is fully asynchronous, since a worker will immediately receive the latest parameter from the server after sending a gradient to the server, without waiting for other workers. Meanwhile, from the perspective of server, BASGD is semi-asynchronous because the server will not update the model until all buffers are filled. However, it is a necessity to limit the updating frequency in ABL when server has no instances. If the server always updates the model when receiving a gradient, it will be easily foiled when Byzantine workers send gradients much more frequently than others. A similar conclusion has been proved in previous works (Damaskinos et al., 2018).

3.2. Aggregation Function

When a SGD step is ready to be executed, there are B buffers providing candidate gradients. An aggregation function is needed to get the final gradient for updating. A naive way is to take the mean of all candidate gradients. However, mean value is sensitive to outliers which are common in BL. For designing proper aggregation functions, we first define the q-Byzantine Robust (q-BR) condition to quantitatively describe the Byzantine resilience ability of an aggregation function.

Definition 1 (q-Byzantine Robust)**.** *For an aggregation function $Aggr(\cdot)$: $Aggr([\mathbf{h}_1, \dots, \mathbf{h}_B]) = \mathbf{G}$, where $\mathbf{G} = [G_1, \dots, G_d]^T$ and $\mathbf{h}_b = [h_{b1}, \dots, h_{bd}]^T, \forall b \in [B]$, we call $Aggr(\cdot)$* q-Byzantine Robust *($q \in \mathbb{Z}, 0 < q < B/2$), if it satisfies the following two properties:*

(a). $Aggr([\mathbf{h}_1+\mathbf{h}', \dots, \mathbf{h}_B+\mathbf{h}']) = Aggr([\mathbf{h}_1, \dots, \mathbf{h}_B])+\mathbf{h}', \ \forall \mathbf{h}_1, \dots, \mathbf{h}_B \in \mathbb{R}^d, \forall \mathbf{h}' \in \mathbb{R}^d$;

(b). $\min_{s\in\mathcal{S}}\{h_{sj}\} \le G_j \le \max_{s\in\mathcal{S}}\{h_{sj}\}, \ \forall j \in [d], \forall \mathcal{S} \subset [B]$ *with* $|\mathcal{S}| = B - q$.

Intuitively, property (a) in Definition 1 says that if all candidate gradients $\mathbf{h}_i$ are added by a same vector $\mathbf{h}'$, the aggregated gradient will also be added by $\mathbf{h}'$. Property (b) says that for each coordinate j, the aggregated value G_j will be between the $(q+1)$-th smallest value and the $(q+1)$-th largest value among the j-th coordinates of all candidate gradients. Thus, the gradient aggregated by a q-BR function is insensitive to at least q outliers. We can find that q-BR condition gets stronger when q increases. Namely, if $Aggr(\cdot)$ is q-BR, then for any $0 < q' < q$, $Aggr(\cdot)$ is also q'-BR.

Remark 1. *When $B > 1$, mean function is not q-Byzantine Robust for any $q > 0$. We illustrate this by a one-dimension example: $h_1, \dots, h_{B-1} \in [0, 1]$, and $h_B = 10 \times B$. Then $\frac{1}{B}\sum_{b=1}^{B} h_b \ge \frac{h_B}{B} = 10 \notin [0, 1]$. Namely, the mean is larger than any of the first $B - 1$ values.*

The following two aggregation functions are both q-BR.

Definition 2 (Coordinate-wise median (Yin et al., 2018))**.** *For candidate gradients $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_B \in \mathbb{R}^d$, $\mathbf{h}_b = [h_{b1}, h_{b2}, \dots, h_{bd}]^T, \ \forall b = 1, 2, \dots, B$.* Coordinate-wise median *is defined as:*

$$Med([\mathbf{h}_1, \dots, \mathbf{h}_B]) = [Med(h_{\cdot 1}), \dots, Med(h_{\cdot d})]^T,$$

where $Med(h_{\cdot j})$ is the scalar median of the j-th coordinates, $\forall j = 1, 2, \dots, d$.

Definition 3 (Coordinate-wise q-trimmed-mean (Yin et al., 2018))**.** *For any positive interger $q < B/2$ and candidate gradients $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_B \in \mathbb{R}^d$, $\mathbf{h}_b = [h_{b1}, h_{b2}, \dots, h_{bd}]^T$, $\forall b = 1, 2, \dots, B$.* Coordinate-wise q-trimmed-mean *is defined as:*

$$Trm([\mathbf{h}_1, \dots, \mathbf{h}_B]) = [Trm(h_{\cdot 1}), \dots, Trm(h_{\cdot d})]^T,$$

where $Trm(h_{\cdot j}) = \frac{1}{B-2q}\sum_{b\in\mathcal{M}_j} h_{bj}$ is the scalar q-trimmed-mean. $\mathcal{M}_j$ is the subset of $\{h_{bj}\}_{b=1}^{B}$ obtained by removing the q largest elements and q smallest elements.

In the following content, coordinate-wise median and coordinate-wise q-trimmed-mean are also called *median* and *trmean*, respectively. Proposition 1 shows the q-BR property of these two functions.

Proposition 1. *Coordinate-wise q-trmean is q-BR, and coordinate-wise median is $\lfloor\frac{B-1}{2}\rfloor$-BR.*

Here, $\lfloor x \rfloor$ represents the maximum integer that is not larger than x. According to Proposition 1, both median and trmean are proper choices for aggregation function in BASGD. The proof can be found in Appendix B of supplementary materials. Now we define another class of aggregation functions, which is also important in analysis in Section 4.

Definition 4 (Stable aggregation function)**.** *Aggregation function $Aggr(\cdot)$ is called* stable *provided that $\forall \mathbf{h}_1, \dots, \mathbf{h}_B$, $\tilde{\mathbf{h}}_1, \dots, \tilde{\mathbf{h}}_B \in \mathbb{R}^d$, letting $\delta = (\sum_{b=1}^{B} \|\mathbf{h}_b - \tilde{\mathbf{h}}_b\|^2)^{\frac{1}{2}}$, we have:*

$$\|Aggr(\mathbf{h}_1, \dots, \mathbf{h}_B) - Aggr(\tilde{\mathbf{h}}_1, \dots, \tilde{\mathbf{h}}_B)\| \le \delta.$$

If $Aggr(\cdot)$ is a stable aggregation function, it means that when there is a disturbance with L_2-norm δ on buffers, the disturbance of aggregated result will not be larger than δ.

Definition 5 (Effective aggregation function)**.** *When there are at most r Byzantine workers, stable aggregation function $Aggr(\cdot)$ is called an (A_1, A_2)-effective aggregation function, provided that it satisfies the following two properties for all $\mathbf{w}^t \in \mathbb{R}^d$ in cases without delay ($\tau_k^t = 0, \ \forall t = 0, 1, \dots, T-1$):*

(i). $\mathbb{E}[\nabla F(\mathbf{w}^t)^T \mathbf{G}_{syn}^t \mid \mathbf{w}^t] \geq \|\nabla F(\mathbf{w}^t)\|^2 - A_1$*;*

(ii). $\mathbb{E}[\|\mathbf{G}_{syn}^t\|^2 \mid \mathbf{w}^t] \leq (A_2)^2$*;*

where $A_1, A_2 \in \mathbb{R}_+$ *are two non-negative constants,* $\mathbf{G}_{syn}^t$ *is the gradient aggregated by* $Aggr(\cdot)$ *at the* t*-th iteration in cases without delay.*

For different aggregation functions, constants A_1 and A_2 may differ. A_1 and A_2 are related to loss function $F(\cdot)$, distribution of instances, buffer number B, maximum Byzantine worker number r. Inequalities (i) and (ii) in Definition 5 are two important properties in convergence proof of synchronous Byzantine learning methods. As revealed in (Yang et al., 2020), there are many existing aggregation rules for Byzantine learning. We find that most of them satisfy Definition 5. For example, Krum, median, and trimmed-mean have already been proved to satisfy these two properties (Blanchard et al., 2017; Yin et al., 2018). SignSGD (Bernstein et al., 2019) can be seen as a combination of 1-bit quantization and median aggregation, while median satisfies the properties in Definition 5.

Compared to Definition 1, Definition 5 can be used to obtain a tighter bound with respect to A_1 and A_2. However, it usually requires more effort to check the two properties in Definition 5 than those in Definition 1.

Please note that too large B could lower the updating frequency and damage the performance, while too small B may harm the Byzantine-resilience. Thus, a moderate B is usually preferred. In some practical applications, we could estimate the maximum number of Byzantine workers r, and set B to make the aggregation function resilient to up to r Byzantine workers. In particular, B is suggested to be $(2r+1)$ for median, since median is $\lfloor \frac{B-1}{2} \rfloor$-BR.

3.3. Mapping Table

At each iteration of BASGD, buffer_b needs at least one gradient for aggregation. In the worst case, all the workers corresponding to buffer_b may be unresponsive. In this case, buffer_b will become the straggler, and slow down the whole learning process. To deal with this problem, we introduce the mapping table for buffer reassignment technique.

We call a worker active worker if it has responsed at the current iteration. If SGD step has not been excuted for Δ seconds, the server immediately zeroes out stored gradients in all buffers, equally reassigns active workers to each buffer, and then continues the learning procedure. Hyper-parameter Δ is called reassignment interval. Figure 2 illustrates an example of reassignment. The grey circles represents unresponsive workers. After reassignment, there are at least one active worker corresponding to each buffer.

Specifically, we introduce a mapping table $\{\beta_s\}_{s=0}^{m-1}$ for buffer reassignment. Initially, $\beta_s = s$ $(\forall s = 0, 1, \ldots, m-1)$. When reassigning buffers, the server only needs to modify the mapping table $\{\beta_s\}_{s=0}^{m-1}$, and then stores worker_s's gradients in buffer_$\{\beta_s \bmod B\}$, instead of buffer_$\{s \bmod B\}$ any more. Please note that the server only needs to modify the mapping table for buffer reassignment, and there is no need to notify workers.

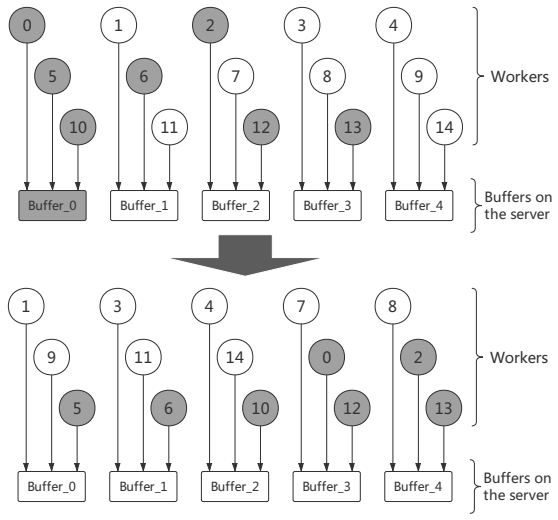


Figure 2. An example of buffer reassignment. White circle represents active worker, and grey circle represents unresponsive worker. Before reassignment, buffer_0 is a straggler. After reassignment, there are at least one active worker corresponding to each buffer.

Besides, a timer is used on the server for indicating when to reassign buffers. The timer is started at the beginning of BASGD, and is restarted immediately after each SGD step or buffer reassignment. When the timer exceeds Δ seconds, buffers will be zeroed out, and reassignment executed. Hyper-parameter Δ should be set properly. If Δ is too small, buffers will be zeroed out too frequently, which may slow down the learning process. If Δ is too large, straggler buffers could not be eliminated in time.

4. Convergence

In this section, we theoretically prove the convergence and resilience of BASGD against failure or attack. There are two main theorems. The first theorem presents a relatively loose but general bound for all q-BR aggregation functions. The second one presents a relatively tight bound for each distinct (A_1, A_2)-effective aggregation function. Since the definition of (A_1, A_2)-effective aggregation function is usually more difficult to verify than q-BR property, the general bound is also useful. Here we only present the results. Proof details are in Appendix B of supplementary materials.

We first make the following assumptions, which have been widely used in stochastic optimization.

Assumption 1 (Lower bound)**.** *Global loss function $F(\mathbf{w})$ is bounded below: $\exists F^* \in \mathbb{R}, F(\mathbf{w}) \geq F^*, \forall \mathbf{w} \in \mathbb{R}^d$.*

Assumption 2 (Bounded bias)**.** *For any loyal worker, it can use locally stored training instances to estimate global gradient with bounded bias κ: $\|\mathbb{E}[\nabla f(\mathbf{w}; z_i)] - \nabla F(\mathbf{w})\| \leq \kappa,\ \forall \mathbf{w} \in \mathbb{R}^d$.*

Assumption 3 (Bounded gradient)**.** *$\nabla F(\mathbf{w})$ is bounded: $\exists D \in \mathbb{R}^+, \|\nabla F(\mathbf{w})\| \leq D,\ \forall \mathbf{w} \in \mathbb{R}^d$.*

Assumption 4 (Bounded variance)**.** $\mathbb{E}[\|\nabla f(\mathbf{w}; z_i) - \mathbb{E}[\nabla f(\mathbf{w}; z_i) \mid \mathbf{w}]\|^2 \mid \mathbf{w}] \leq \sigma^2,\ \forall \mathbf{w} \in \mathbb{R}^d$.

Assumption 5 (L-smoothness)**.** *Global loss function $F(\mathbf{w})$ is differentiable and L-smooth: $||\nabla F(\mathbf{w}) - \nabla F(\mathbf{w}')|| \leq L||\mathbf{w} - \mathbf{w}'||,\ \forall \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$.*

Let $N^{(t)}$ be the $(q+1)$-th smallest value in $\{N_b^t\}_{b\in[B]}$, where N_b^t is the number of gradients stored in buffer b at the t-th iteration. We define the constant

$$\Lambda_{B,q,r} = \frac{(B-r)\sqrt{B-r+1}}{\sqrt{(B-q-1)(q-r+1)}},$$

which will appear in Lemma 1 and Lemma 2.

Lemma 1. *If $Aggr(\cdot)$ is q-BR, and there are at most r Byzantine workers ($r \leq q$), we have:*

$$\mathbb{E}[||\mathbf{G}^t||^2 \mid \mathbf{w}^t] \leq \Lambda_{B,q,r} d \cdot (D^2 + \sigma^2/N^{(t)}).$$

Lemma 2. *If $Aggr(\cdot)$ is q-BR, and the total number of heavily delayed workers and Byzantine workers is not larger than r ($r \leq q$), we have:*

$$\begin{aligned}&||\mathbb{E}[\mathbf{G}^t - \nabla F(\mathbf{w}^t) \mid \mathbf{w}^t]||\\ \leq&\Lambda_{B,q,r} d(\tau_{max} L \cdot [\Lambda_{B,q,r} d(D^2 + \sigma^2/N^{(t)})]^{\frac{1}{2}} + \sigma + \kappa).\end{aligned}$$

Theorem 1. *Let $\tilde{D} = \frac{1}{T}\sum_{t=0}^{T-1}(D^2 + \sigma^2/N^{(t)})^{\frac{1}{2}}$. If $Aggr(\cdot)$ is q-BR, $B = O(r)$, and the total number of heavily delayed workers and Byzantine workers is not larger than r ($r \leq q$), with learning rate $\eta = O(\frac{1}{L\sqrt{T}})$, we have:*

$$\begin{aligned}\frac{\sum_{t=0}^{T-1}\mathbb{E}[||\nabla F(\mathbf{w}^t)||^2]}{T} \leq& O\left(\frac{L[F(\mathbf{w}^0) - F^*]}{T^{\frac{1}{2}}}\right)\\ &+ O\left(\frac{rd\tilde{D}}{T^{\frac{1}{2}}(q-r+1)^{\frac{1}{2}}}\right) + O\left(\frac{rDd\sigma}{(q-r+1)^{\frac{1}{2}}}\right)\\ &+ O\left(\frac{rDd\kappa}{(q-r+1)^{\frac{1}{2}}}\right) + O\left(\frac{r^{\frac{3}{2}}LD\tilde{D}d^{\frac{3}{2}}\tau_{max}}{(q-r+1)^{\frac{3}{4}}}\right).\end{aligned}$$

Please note that the convergence rate of vanilla ASGD is $O(T^{-\frac{1}{2}})$. Hence, Theorem 1 indicates that BASGD has a theoretical convergence rate as fast as vanilla ASGD, with an extra constant variance. The term $O(rDd\sigma(q-r+1)^{-\frac{1}{2}})$ is caused by the aggregation function, which can be deemed as a sacrifice for Byzantine resilience. The term $O(rDd\kappa(q-r+1)^{-\frac{1}{2}})$ is caused by the differences of training instances among different workers. In independent and identically distributed (i.i.d.) cases, $\kappa = 0$ and the term vanishes. The term $O(r^{\frac{3}{2}}LD\tilde{D}d^{\frac{3}{2}}\tau_{max}(q-r+1)^{-\frac{3}{4}})$ is caused by the delay, and related to parameter τ_{max}. The term is also related to the buffer size. When N_b^t increases, $N^{(t)}$ may increase, and thus $\tilde{D}$ will decrease. Namely, larger buffer size will result in smaller $\tilde{D}$. Besides, the factor $(q-r+1)^{-\frac{1}{2}}$ or $(q-r+1)^{-\frac{3}{4}}$ decreases as q increases, and increases as r increases.

Although general, the bound presented in Theorem 1 is relatively loose in high-dimensional cases, since d appears in all the three extra terms. To obtain a tighter bound, we introduce Theorem 2 for BASGD with (A_1, A_2)-effective aggregation function (Definition 5).

Theorem 2. *If the total number of heavily delayed workers and Byzantine workers is not larger than r, $B = O(r)$, and $Aggr(\cdot)$ is an (A_1, A_2)-effective aggregation function in this case. With learning rate $\eta = O(\frac{1}{\sqrt{LT}})$, in general asynchronous cases we have:*

$$\begin{aligned}\frac{\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla F(\mathbf{w}^t)\|^2]}{T} \leq& O\left(\frac{L^{\frac{1}{2}}[F(\mathbf{w}^0) - F^*]}{T^{\frac{1}{2}}}\right)\\ &+ O\left(\frac{L^{\frac{1}{2}}\tau_{max}DA_2r^{\frac{1}{2}}}{T^{\frac{1}{2}}}\right) + O\left(\frac{L^{\frac{1}{2}}(A_2)^2}{T^{\frac{1}{2}}}\right)\\ &+ O\left(\frac{L^{\frac{5}{2}}(A_2)^2\tau_{max}^2 r}{T^{\frac{3}{2}}}\right) + A_1.\end{aligned}$$

Theorem 2 indicates that if $Aggr(\cdot)$ makes a synchronous BL method converge (i.e., satisfies Definition 5), BASGD converges when using $Aggr(\cdot)$ as aggregation function. Hence, BASGD can also be seen as a technique of asynchronization. That is to say, new asynchronous methods can be obtained from synchronous ones when using BASGD. The extra constant term A_1 is caused by gradient bias. When there is no Byzantine workers ($r = 0$), and instances are i.i.d. across workers, letting $B = 1$ and $Aggr(\mathbf{h}_1, \ldots, \mathbf{h}_B) = Aggr(\mathbf{h}_1) = \mathbf{h}_1$, BASGD degenerates to vanilla ASGD. In this case, there is no gradient bias ($A_1 = 0$), and BASGD has a convergence rate of $O(1/\sqrt{T})$, which is the same as that of vanilla ASGD (Liu & Zhang, 2021).

Meanwhile, it remains uncertain whether the dependence to the staleness parameter τ_{max} is tight enough. Theorem 2 illustrates that BASGD has a convergence rate of $O(\tau_{max}/T^{\frac{1}{2}})$, while the convergence rate of vanilla ASGD can reach $O(\tau_{max}/T)$. To the best of our knowledge, there

exist few works revealing the tightness of τ_{max} in asynchronous BL, and we will leave this for future work.

In general cases, Theorem 2 guarantees BASGD to find a point such that the squared L_2-norm of its gradient is not larger than A_1 (but not necessarily around a stationary point), in expectation. Please note that Assumption 3 already guarantees that gradient's squared L_2-norm is not larger than D^2. We introduce Proposition 2 to show that A_1 is guaranteed to be smaller than D^2 under a mild condition.

Proposition 2. *$Aggr(\cdot)$ is an (A_1, A_2)-effective aggregation function, and $\mathbf{G}_{syn}^t$ is aggregated by $Aggr(\cdot)$ in synchronous setting. If $\mathbb{E}[\|\mathbf{G}_{syn}^t - \nabla F(\mathbf{w}^t)\| \mid \mathbf{w}^t] \leq D$, $\forall \mathbf{w}^t \in \mathbb{R}^d$, then we have $A_1 \leq D^2$.*

$\mathbf{G}_{syn}^t$ is the aggregated result of $Aggr(\cdot)$, and is a robust estimator of $\nabla F(\mathbf{w}^t)$ used for updating. Since $\|\nabla F(\mathbf{w}^t)\| \leq D$, $\nabla F(\mathbf{w}^t)$ locates in a ball with radius D. $\mathbb{E}[\|\mathbf{G}_{syn}^t - \nabla F(\mathbf{w}^t)\| \mid \mathbf{w}^t] \leq D$ means that the bias of $\mathbf{G}_{syn}^t$ is not larger than the radius D, which is a mild condition for $Aggr(\cdot)$.

As many existing works have shown (Assran et al., 2020; Nokleby et al., 2020), speed-up is also an important aspect of distributed learning methods. In BASGD, different workers can compute gradients concurrently, make each buffer be filled more quickly, and thus speed up the model updating. However, we mainly focus on Byzantine-resilience in this work. Speed-up will be thoroughly studied in future work.

5. Experiment

In this section, we empirically evaluate the performance of BASGD and baselines in both image classification (IC) and natural language processing (NLP) applications. Our experiments are conducted on a distributed platform with dockers. Each docker is bound to an NVIDIA Tesla V100 (32G) GPU (in IC) or an NVIDIA Tesla K80 GPU (in NLP). Please note that different GPU cards do not affect the reported metrics in the experiment. We choose 30 dockers as workers in IC, and 8 dockers in NLP. An extra docker is chosen as server. All algorithms are implemented with PyTorch 1.3.

5.1. Experimental Setting

We compare the performance of different methods under two types of attack: negative gradient attack (NG-attack) and random disturbance attack (RD-attack). Byzantine workers with NG-attack send $\tilde{\mathbf{g}}_{NG} = -k_{atk} \cdot \mathbf{g}$ to server, where $\mathbf{g}$ is the true gradient and $k_{atk} \in \mathbb{R}_+$ is a parameter. Byzantine workers with RD-attack send $\tilde{\mathbf{g}}_{RD} = \mathbf{g} + \mathbf{g}_{rnd}$ to server, where $\mathbf{g}_{rnd}$ is a random vector sampled from normal distribution $\mathcal{N}(\mathbf{0}, \|\sigma_{atk}\mathbf{g}\|^2 \cdot \mathbf{I})$. Here, σ_{atk} is a parameter and $\mathbf{I}$ is an identity matrix. NG-attack is a typical kind of malicious attack, while RD-attack can be seen as an accidental failure with expectation $\mathbf{0}$. Besides, each worker is manually set to have a delay, which is k_{del} times the computing time. Training set is randomly and equally distributed to different workers. We use the average top-1 test accuracy (in IC) or average perplexity (in NLP) on all workers w.r.t. epochs as final metrics. For BASGD, we use median and trimmed-mean as aggregation function. Reassignment interval is set to be 1 second. Top-1 test accuracy (in IC) w.r.t. wall-clock time of BASGD with more aggregation functions is reported in Appendix C of supplementary material.

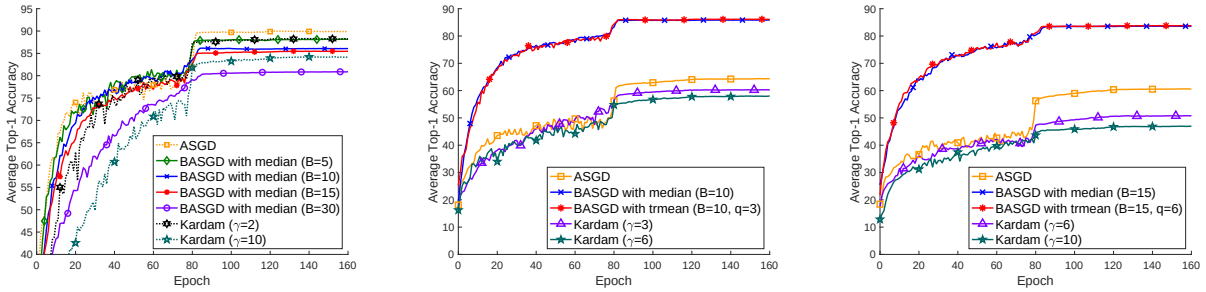
Because BASGD is an ABL method, SBL methods cannot be directly compared with BASGD. The ABL method Zeno++ either cannot be directly compared with BASGD, because Zeno++ needs to store some instances on server. The number of instances stored on server will affect the performance of Zeno++ (Xie et al., 2020). Hence, we compare BASGD with ASGD and Kardam in our experiments. We set dampening function $\Lambda(\tau) = \frac{1}{1+\tau}$ for Kardam as suggested in (Damaskinos et al., 2018).

5.2. Image Classification Experiment

In IC experiment, algorithms are evaluated on CIFAR-10 (Krizhevsky et al., 2009) with deep learning model ResNet-20 (He et al., 2016). Cross-entropy is used as the loss function. We set $k_{atk} = 10$ for NG-attack, and $\sigma_{atk} = 0.2$ for RD-attack. k_{del} is randomly sampled from truncated standard normal distribution within $[0, +\infty)$. As suggested in (He et al., 2016), learning rate η is set to 0.1 initially for each algorithm, and multiplied by 0.1 at the 80-th epoch and the 120-th epoch respectively. The weight decay is set to 10^{-4}. We run each algorithm for 160 epochs. Batch size is set to 25.

Firstly, we compare the performance of different methods when there are no Byzantine workers. Experimental results with median and trmean aggregation functions are illustrated in Figure 3(a) and Figure 3(d), respectively. ASGD achieves the best performance. BASGD ($B > 1$) and Kardam have similar convergence rate to ASGD, but both sacrifice a little accuracy. Besides, the performance of BASGD gets worse when the buffer number B increases, which is consistent with the theoretical results. Please note that ASGD is a degenerated case of BASGD when $B = 1$ and $Aggr(\mathbf{h}_1) = \mathbf{h}_1$. Hence, BASGD can achieve the same performance as ASGD when there is no failure or attack.

Then, for each type of attack, we conduct two experiments in which there are 3 and 6 Byzantine workers, respectively. We respectively set 10 and 15 buffers for BASGD in these two experiments. For space saving, we only present average top-1 test accuracy in Figure 3(b) and Figure 3(e) (3 Byzantine workers), and Figure 3(c) and Figure 3(f) (6 Byzantine workers). Results about training loss are in Appendix C. We can find that BASGD significantly outperforms ASGD and



(a) no attack (b) 3 Byzantine workers (RD-attack) (c) 6 Byzantine workers (RD-attack)

(d) no attack (e) 3 Byzantine workers (NG-attack) (f) 6 Byzantine workers (NG-attack)

Figure 3. Average top-1 test accuracy w.r.t. epochs when there are no Byzantine workers (left column), 3 Byzantine workers (middle column) and 6 Byzantine workers (right column), respectively. Subfigures (b) and (c) are for RD-attack, while (e) and (f) for NG-attack.

Table 1. Filtered ratio of received gradients in Kardam under NG-attack in IC task (3 Byzantine workers)

TERM	BY FREQUENCY FILTER	BY LIPSCHITZ FILTER	IN TOTAL
LOYAL GRADS ($\gamma = 3$)	10.15% (31202/307530)	40.97% (126000/307530)	51.12%
BYZANTINE GRADS ($\gamma = 3$)	10.77% (3681/34170)	40.31% (13773/34170)	51.08%
LOYAL GRADS ($\gamma = 8$)	28.28% (86957/307530)	28.26% (86893/307530)	56.53%
BYZANTINE GRADS ($\gamma = 8$)	28.38% (9699/34170)	28.06% (9588/34170)	56.44%
LOYAL GRADS ($\gamma = 14$)	85.13% (261789/307530)	3.94% (12117/307530)	89.07%
BYZANTINE GRADS ($\gamma = 14$)	84.83% (28985/34170)	4.26% (1455/34170)	89.08%

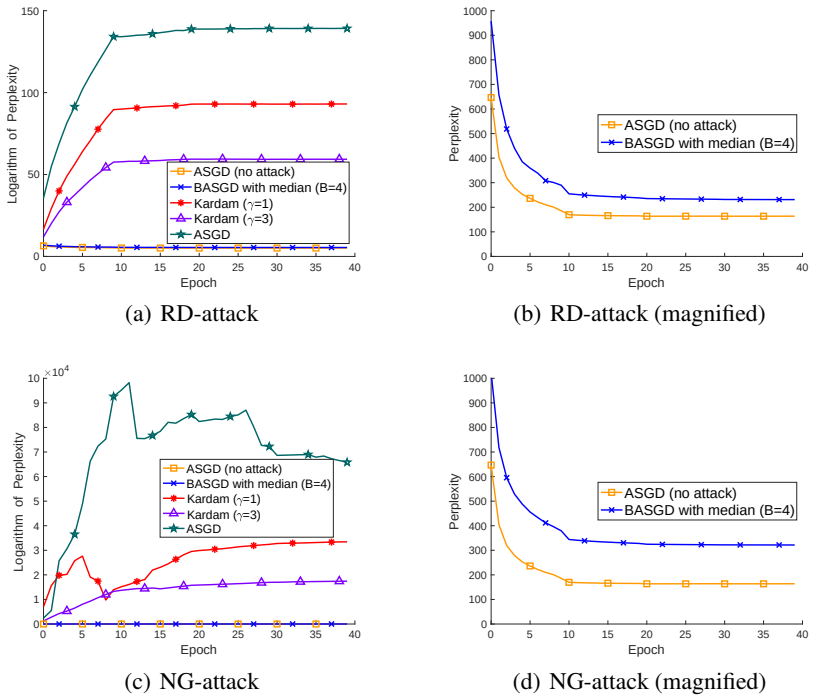
Kardam under both RD-attack (accidental failure) and NG-attack (malicious attack). Under the less harmful RD-attack, although ASGD and Kardam still converge, they both suffer a significant loss on accuracy. Under NG-attack, both ASGD and Kardam cannot converge, even if we have tried different values of *assumed Byzantine worker number* for Kardam, which is denoted by a hyper-parameter γ in this paper. Hence, both ASGD and Kardam cannot resist malicious attack. On the contrary, BASGD still has a relatively good performance under both types of attack.

Moreover, we count the ratio of filtered gradients in Kardam, which is shown in Table 1. We can find that in order to filter Byzantine gradients, Kardam also filters approximately equal ratio of loyal gradients. It explains why Kardam performs poorly under malicious attack.

5.3. Natural Language Processing Experiment

In NLP experiment, the algorithms are evaluated on the WikiText-2 dataset with LSTM (Hochreiter & Schmidhuber, 1997) networks. We only use the training set and test set, while the validation set is not used in our experiment. For LSTM, we adopt 2 layers with 100 units in each. Word embedding size is set to 100, and sequence length is set to 35. Gradient clipping size is set to 0.25. Cross-entropy is used as the loss function. For each algorithm, we run each algorithm for 40 epochs. Initial learning rate η is chosen from $\{1, 2, 5, 10, 20\}$, and is divided by 4 every 10 epochs. The best test result is adopted as the final one.

The performance of ASGD under no attack is used as gold standard. We set $k_{atk} = 10$ and $\sigma_{atk} = 0.1$. One of the eight workers is Byzantine. k_{del} is randomly sampled from



(a) RD-attack (b) RD-attack (magnified)

(c) NG-attack (d) NG-attack (magnified)

Figure 4. Average perplexity w.r.t. epochs with 1 Byzantine worker. Subfigures (a) and (b) are for RD-attack, while Subfigures (c) and (d) for NG-attack. Due to the differences in magnitude of perplexity, y-axes of Subfigures (a) and (c) are in log-scale. In addition, Subfigures (b) and (d) illustrate that BASGD converges with only a little loss in perplexity compared to the gold standard.

exponential distribution with parameter $\lambda = 1$. Each experiment is carried out for 3 times, and the average perplexity is reported in Figure 4. We can find that BASGD converges under each kind of attack, with only a little loss in perplexity compared to the gold standard (ASGD without attack). On the other hand, ASGD and Kardam both fail, even if we have set the largest γ ($\gamma = 3$) for Kardam.

6. Conclusion

In this paper, we propose a novel method called BASGD for asynchronous Byzantine learning. To the best of our knowledge, BASGD is the first ABL method that can resist malicious attack without storing any instances on server. Compared with those methods which need to store instances on server, BASGD has a wider scope of application. BASGD is proved to be convergent, and be able to resist failure or attack. Empirical results show that BASGD significantly outperforms vanilla ASGD and other ABL baselines, when there exists failure or attack on workers.

Acknowledgements

This work is supported by National Key R&D Program of China (No. 2020YFA0713900), NSFC-NRF Joint Research Project (No. 61861146001) and NSFC Project (No. 61921006).

References

Alistarh, D., Allen-Zhu, Z., and Li, J. Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 4613–4623, 2018.

Assran, B. M., Aytekin, A., Feyzmahdavian, H. R., Johansson, M., and Rabbat, M. G. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

Baruch, G., Baruch, M., and Goldberg, Y. A little is enough: Circumventing defenses for distributed learning. In *Advances in Neural Information Processing Systems*, pp. 8635–8645, 2019.

Bernstein, J., Zhao, J., Azizzadenesheli, K., and Anandkumar, A. signSGD with majority vote is communication efficient and fault tolerant. In *Proceedings of the International Conference on Learning Representations*, 2019.

Blanchard, P., Guerraoui, R., Stainer, J., et al. Machine learning with adversaries: Byzantine tolerant gradient

descent. In *Advances in Neural Information Processing Systems*, pp. 119–129, 2017.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the International Conference on Computational Statistics*, pp. 177–186. Springer, 2010.

Chen, Y., Su, L., and Xu, J. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, 2017.

Damaskinos, G., Guerraoui, R., Patra, R., Taziki, M., et al. Asynchronous Byzantine machine learning (the case of SGD). In *Proceedings of the International Conference on Machine Learning*, pp. 1145–1154, 2018.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

Diakonikolas, I. and Kane, D. M. Recent advances in algorithmic high-dimensional robust statistics. *arXiv preprint arXiv:1911.05911*, 2019.

Diakonikolas, I., Kamath, G., Kane, D. M., Li, J., Moitra, A., and Stewart, A. Being robust (in high dimensions) can be practical. In *Proceedings of the International Conference on Machine Learning*, pp. 999–1008, 2017.

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

Haddadpour, F., Kamani, M. M., Mahdavi, M., and Cadambe, V. Trading redundancy for communication: Speeding up distributed SGD for non-convex optimization. In *Proceedings of the International Conference on Machine Learning*, pp. 2545–2554, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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

Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M. I. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pp. 3068–3076, 2014.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pp. 315–323, 2013.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *arXiv:1912.04977*, 2019.

Konevcnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv:1610.05492*, 2016.

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

Lee, J. D., Lin, Q., Ma, T., and Yang, T. Distributed stochastic variance reduced gradient methods by sampling extra data with replacement. *The Journal of Machine Learning Research*, 18(1):4404–4446, 2017.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pp. 19–27, 2014.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

Lin, Q., Lu, Z., and Xiao, L. An accelerated proximal coordinate gradient method. In *Advances in Neural Information Processing Systems*, pp. 3059–3067, 2014.

Liu, J. and Zhang, C. Distributed learning systems with first-order methods. *arXiv preprint arXiv:2104.05245*, 2021.

Ma, C., Smith, V., Jaggi, M., Jordan, M., Richtárik, P., and Takác, M. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the International Conference on Machine Learning*, pp. 1973–1982, 2015.

Nokleby, M., Raja, H., and Bajwa, W. U. Scaling-up distributed processing of data streams for machine learning. *arXiv preprint arXiv:2005.08854*, 2020.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Shalev-Shwartz, S. and Zhang, T. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

Shamir, O., Srebro, N., and Zhang, T. Communication-efficient distributed optimization using an approximate newton-type method. In *Proceedings of the International Conference on Machine Learning*, pp. 1000–1008, 2014.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

Sun, S., Chen, W., Bian, J., Liu, X., and Liu, T.-Y. Slim-dp: a multi-agent system for communication-efficient distributed deep learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 721–729, 2018.

Wangni, J., Wang, J., Liu, J., and Zhang, T. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pp. 1299–1309, 2018.

Xiao, L. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596, 2010.

Xie, C., Koyejo, S., and Gupta, I. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *Proceedings of the International Conference on Machine Learning*, pp. 6893–6901, 2019.

Xie, C., Koyejo, S., and Gupta, I. Zeno++: Robust fully asynchronous SGD. In *Proceedings of the International Conference on Machine Learning*, 2020.

Yang, T. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pp. 629–637, 2013.

Yang, Z., Gang, A., and Bajwa, W. U. Adversary-resilient distributed and decentralized statistical inference and machine learning: An overview of recent advances under the byzantine threat model. *IEEE Signal Processing Magazine*, 37(3):146–159, 2020.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the International Conference on Machine Learning*, pp. 5650–5659, 2018.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. Defending against saddle point attack in byzantine-robust distributed learning. In *Proceedings of the International Conference on Machine Learning*, pp. 7074–7084, 2019.

Yu, H., Jin, R., and Yang, S. On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization. In *Proceedings of the International Conference on Machine Learning*, pp. 7184–7193, 2019a.

Yu, H., Yang, S., and Zhu, S. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5693–5700, 2019b.

Zhang, L., Mahdavi, M., and Jin, R. Linear convergence with condition number independent access of full gradients. In *Advances in Neural Information Processing Systems*, pp. 980–988, 2013.

Zhang, R. and Kwok, J. Asynchronous distributed admm for consensus optimization. In *Proceedings of the International Conference on Machine Learning*, pp. 1701–1709, 2014.

Zhao, S.-Y., Xiang, R., Shi, Y.-H., Gao, P., and Li, W.-J. SCOPE: scalable composite optimization for learning on spark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2928–2934. AAAI Press, 2017.

Zhao, S.-Y., Zhang, G.-D., Li, M.-W., and Li, W.-J. Proximal SCOPE for distributed sparse learning. In *Advances in Neural Information Processing Systems*, pp. 6551–6560, 2018.

Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.-M., and Liu, T.-Y. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the International Conference on Machine Learning*, pp. 4120–4129, 2017.

Zhou, Y., Liang, Y., Yu, Y., Dai, W., and Xing, E. P. Distributed proximal gradient algorithm for partially asynchronous computer clusters. *The Journal of Machine Learning Research*, 19(1):733–764, 2018.

Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 2595–2603, 2010.