Asynchronous Byzantine Machine Learning (the case of SGD)

Georgios Damaskinos [1] **El Mahdi El Mhamdi** [1] **Rachid Guerraoui** [1] **Rhicheek Patra** [1] **Mahsa Taziki** [1]

Abstract

Asynchronous distributed machine learning solutions have proven very effective so far, but always assuming perfectly functioning workers. In practice, some of the workers can however exhibit Byzantine behavior, caused by hardware failures, software bugs, corrupt data, or even malicious attacks. We introduce *Kardam*, the first distributed asynchronous stochastic gradient descent (SGD) algorithm that copes with Byzantine workers. Kardam consists of two complementary components: a filtering and a dampening component. The first is scalar-based and ensures resilience against $\frac{1}{3}$ Byzantine workers. Essentially, this filter leverages the Lipschitzness of cost functions and acts as a self-stabilizer against Byzantine workers that would attempt to corrupt the progress of SGD. The dampening component bounds the convergence rate by adjusting to stale information through a generic gradient weighting scheme. We prove that Kardam guarantees almost sure convergence in the presence of asynchrony and Byzantine behavior, and we derive its convergence rate. We evaluate Kardam on the CIFAR-100 and EMNIST datasets and measure its overhead with respect to non Byzantine-resilient solutions. We empirically show that Kardam does not introduce additional noise to the learning procedure but does induce a slowdown (the cost of Byzantine resilience) that we both theoretically and empirically show to be less than f/n, where f is the number of Byzantine failures tolerated and n the total number of workers. Interestingly, we also empirically observe that the dampening component is interesting in its own right for it enables to build an SGD algorithm that outperforms alternative staleness-aware asynchronous competitors in environments with honest workers.

[1]EPFL, Lausanne, Switzerland. Correspondence to: (without spaces) <firstname.lastname@epfl.ch>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

1. Introduction

To keep up with the amount of data available today and the corresponding increasing demand for resources, machine learning (ML) practitioners rely on large scale distributed systems (Dean et al., 2012; Abadi et al., 2016; Li et al., 2013; 2014b;a; Ho et al., 2013; Cui et al., 2016). Most of these systems make use of the same work-horse optimization algorithm: *stochastic gradient descent* (SGD), typically following the parameter server scheme (Li et al., 2014a;b). The computation is divided into *epochs*, i.e., *model* (parameter vector) updates. The server gathers gradients from the workers and employs them to perform a single model update, then broadcasts the new model to every worker for computing new gradients (based on random data samples).

To be practical, a distributed ML solution should not assume that all workers perform perfectly well. Some arbitrary behavior of at least a fraction of the workers should be tolerated. The Byzantine failure model (Lamport et al., 1982) offers an elegant abstraction to reason about problems of adversarial machine learning. In particular, a Byzantine behavior can be due to a crash, a software bug, a stale local view of a model, a corrupt piece of data, or worse, to attackers that benefit from a security flaw in a device and compromise its behavior. The Byzantine model encompasses the problem of *poisoning attacks* (Biggio & Laskov, 2012; Muñoz-González et al., 2017; Kurakin et al., 2016). For instance, a group of adversarial (Byzantine) workers could bias the gradient estimator and prevent convergence by sending corrupt gradients. Byzantine-resilient (or simply Byzantine) ML solutions are very appealing for they do not make any assumption on the behavior of Byzantine workers.

A few Byzantine distributed ML solutions have been recently proposed (Blanchard et al., 2017; Su, 2017; El Mhamdi et al., 2018). All however assume a restrictive synchronous model. In each epoch, (1) all (honest) workers are supposed to use the exact same model to compute the gradient, and (2) the parameter server waits for a quorum of workers before aggregating their gradients. When networks are asynchronous, exhibiting heterogeneous (sometimes arbitrary) communication delays, synchronous solutions inevitably lead to slow convergence.

Asynchronous SGD algorithms, on the other hand, enable huge performance benefits despite heterogeneous communi-

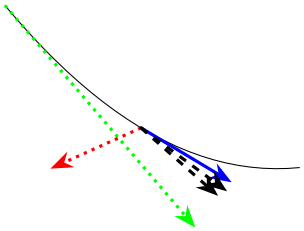


Figure 1: The gradients computed by non-stale honest workers (black dashed arrows) are distributed around (and are on average equal to) the actual gradient (solid blue arrow) of the cost function (thin black curve). A Byzantine worker can propose an arbitrary poisoning vector (red dotted arrow). A honest but stale worker computes the correct gradient but for a stale version of the model (long green dotted arrow).

cation delays (Lian et al., 2016; Liu et al., 2015; Ho et al., 2013; Li et al., 2014a). In short, such algorithms (1) allow workers to make use of a stale model as well as (2) update the model as soon as a new gradient is delivered (instead of waiting for a quorum). Nevertheless, none of these asynchronous algorithms tolerate any Byzantine behavior. In fact, all provably convergent asynchronous SGD algorithms assume that all the workers are permanently honest about their gradient, i.e., provide unbiased estimations of the actual gradient (Figure 1).

Combining asynchrony with Byzantine resilience is challenging. In particular, aggregating gradients that were computed on different models requires the knowledge of how the curvature of the cost function evolves with staleness. This curvature determines the window of synchrony within which a synchronous method can be transposed into an asynchronous context. Roughly speaking, the more locally curved the cost function is, the narrower this window and vice versa. Estimating the curvature requires heavy computations of the Hessian matrix ($\mathcal{O}(d^2)$), not to mention the fact that this would deprive the parameter server from the most prominent advantage of asynchrony, namely updating the model as soon as a *single* gradient is delivered (i.e., the parameter server would need to aggregate a quorum).

In this paper, we consider for the first time the situation where a significant fraction of workers ($\frac{f}{n}$) can be Byzantine (arbitrarily adversarial) and consider unbounded communication delays. Such situation corresponds to that of many realistic distributed platforms today. We present the first asynchronous Byzantine gradient descent algorithm, we call *Kardam*. Kardam leverages the Lipschitzness of the cost function to filter out gradients from potentially Byzantine workers, while prohibiting Byzantine workers from flooding the parameter server (which in turn would prevent honest workers from updating the model). Kardam also uses a dampening scheme that scales each gradient based on its staleness. The computation overhead for each update is negligible as the filtering component of Kardam is mostly scalar-based. The time complexity for each update computed in terms of the dimension d of a gradient is $\mathcal{O}(d + fn)$. This complexity is the same as the standard complexity of an asynchronous SGD update ($\mathcal{O}(d)$) for the very high-dimensional learning models of today ($d \gg (f, n)$). We prove the convergence of Kardam and precisely determine its convergence rate. In particular, we prove its self-stabilizing property using a refined version of the global confinement argument (Bottou, 1998).

We implemented and deployed Kardam in a distributed setting and we report in this paper on its in-depth empirical evaluation on the CIFAR-100 and EMNIST datasets. In particular, we evaluate the overhead of Kardam with respect to non Byzantine-resilient solutions. Kardam does not tamper with the learning procedure (i.e., include additional noise), yet it does induce a slowdown that we empirically show to be less than $\frac{f}{n}$, where f is the number of Byzantine failures tolerated and n the total number of workers (we also prove that theoretically). Finally, we show that the dampening component (when plugged on an asynchronous non Byzantine-resilient SGD solution) outperforms alternative staleness-aware asynchronous competitors in environments with honest workers.

The code to reproduce our experiments as well as a few additional results (varying f) will be found at https://github.com/LPD-EPFL/kardam.

2. Computing Model

We consider the general distributed model for machine learning, namely the parameter server (Dean et al., 2012; Abadi et al., 2016; Li et al., 2013; 2014b;a; Ho et al., 2013; Cui et al., 2016)[1]. We assume that f of the n workers are Byzantine (behave arbitrarily). Following the traditional assumption in distributed computing, we assume that the identities of the Byzantine workers are unknown whereas f (in practice, an upper bound) is known. Computation is divided into (infinitely many) asynchronous model updates (epochs).

Definition 1 (Time)**.** *The global epoch (denoted by t) represents the global logical clock of the parameter server (or equivalently the number of model updates). The local timestamp (denoted by l_p) for a given worker p, represents the epoch of the model that the worker receives from the server and computes the gradient upon. The difference $t - l_p$ can be arbitrarily large due to the asynchrony of the network.*

During each epoch t, the parameter server broadcasts the model $\boldsymbol{x}_t \in \mathbb{R}^d$ to all the workers. A cost function Q reflects the quality of the model for the learning task. Each non-Byzantine worker p computes an estimate $\boldsymbol{g}_p = \boldsymbol{G}(\boldsymbol{x}_{l_p}, \xi_p)$

[1] Classical techniques of state-machine replication (Lynch, 1996) can be used to ensure that the parameter server is reliable.

t	Epoch at the parameter server,incremented after each update.
l_p	Timestamp (given by the parameter server) of the model currently used by worker p.
$\boldsymbol{x}_t$	Model (parameter vector) at epoch t with dimensionality d.
γ_t	Learning rate at epoch t s.t $\sum_{t=1}^{\infty} \gamma_t = \infty$ and $\sum_{t=1}^{\infty} \gamma_t^2 < \infty$.
$\boldsymbol{g}_p$	Each gradient is a tuple $[\boldsymbol{g}_p, l]$ denoting that a worker p computed the gradient $\boldsymbol{g}_p$ w.r.t $\boldsymbol{x}_l$.
$\|X\|$	Cardinality of a set X.
M	Number of gradients that the server waits for before updating the model parameters. ($M = 1$ in asynchrony).
$\mathcal{G}_t$	Set of gradients that the server receives in epoch t. Note that $\|\mathcal{G}_t\| = M$.
τ_{tl}	Staleness value for a gradient $[\boldsymbol{g}, l]$ at epoch t ($\tau_{tl} \triangleq k - l$).
ξ	Mini-batch of training examples.
$Q(\boldsymbol{x})$	Cost function for a model $\boldsymbol{x}$.
K	Global Lipschitz coefficient of $\boldsymbol{\nabla} Q$, i.e $K = \sup_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d} (\frac{\|\boldsymbol{\nabla} Q(\boldsymbol{x}) - \boldsymbol{\nabla} Q(\boldsymbol{y})\|}{\|\boldsymbol{x} - \boldsymbol{y}\|})$.

Table 1: The notations used in Kardam.

of the actual gradient $\boldsymbol{\nabla} Q(\boldsymbol{x}_{l_p})$ of the cost function Q, where ξ_p is a random variable representing, for example, the sample (or a mini-batch of samples) drawn from the dataset at worker p. Each worker p sends the timestamp l_p (to declare which version of the model it used) and the gradient $\boldsymbol{g}_p$. See Table 1 for notational details.

A Byzantine worker b proposes a gradient $\boldsymbol{g}_b$ which can deviate arbitrarily from $\boldsymbol{G}(\boldsymbol{x}_{l_b}, \xi_b)$ (see Figure 1). A Byzantine worker may have full knowledge of the system, including the gradients proposed by other workers. Byzantine workers can furthermore collude, as typically assumed in the distributed computing literature (Lamport et al., 1982; Lynch, 1996; Cachin et al., 2011). Since the communication is assumed to be asynchronous, the parameter server takes into account the first gradient received at time t. The parameter server then either suspects the gradient and ignores it, or employs it to update the model and move to epoch $t+1$. We make the following assumptions about any honest worker p.

Assumption 1 (Unbiased gradient estimator)**.**

$$\mathbb{E}_{\xi_p} \boldsymbol{G}(\boldsymbol{x}_{l_p}, \xi_p) = \boldsymbol{\nabla} Q(\boldsymbol{x}_{l_p})$$

Assumption 2 (Bounded variance)**.**

$$\mathbb{E}_{\xi_p} \|\boldsymbol{G}(\boldsymbol{x}_{l_p}, \xi_p) - \boldsymbol{\nabla} Q(\boldsymbol{x}_{l_p})\|^2 \leq d\sigma^2$$

Assumptions 1 and 2 are common in the literature (Bottou, 1998) and hold if the data used for computing the gradients is drawn uniformly and independently.

Assumption 3 (Linear growth of r-th moment)**.**

$$\mathbb{E}_{\xi_p} \|\boldsymbol{G}(\boldsymbol{x}, \xi_p)\|^r \leq A_r + B_r \|\boldsymbol{x}\|^r \quad \forall \boldsymbol{x} \in \mathbb{R}, \quad r = 2, 3, 4$$

Assumption 3 translates into "the r-th moment of the gradient estimator grows linearly with the r-th power of the norm of the model" as assumed in (Bottou, 1998).

Assumption 4 (Lipschitz gradient)**.**

$$||\boldsymbol{\nabla} Q(\boldsymbol{x}_1) - \boldsymbol{\nabla} Q(\boldsymbol{x}_2)|| \leq K ||\boldsymbol{x}_1 - \boldsymbol{x}_2||$$

Assumption 5 (Convexity in the horizon)**.** *We require that beyond a certain horizon, $\|\boldsymbol{x}\| \geq D$, there exist $\epsilon > 0$ and $0 \leq \beta < \pi/2$ such that $\|\boldsymbol{\nabla} Q(\boldsymbol{x})\| \geq \epsilon > 0$ and $\frac{\langle \boldsymbol{x}, \boldsymbol{\nabla} Q(\boldsymbol{x}) \rangle}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{\nabla} Q(\boldsymbol{x})\|} \geq \cos \beta$.*

Assumptions 4 and 5 are the same as in (Blanchard et al., 2017), the first is classic, the second is a slight refinement of a similar assumption in (Bottou, 1998). It essentially states that, beyond a certain horizon D in the parameter space, the opposite of the gradient points towards the origin.

Definition 2 (Byzantine resilience)**.** *Let Q be any cost function satisfying the assumptions above. Let A be any distributed SGD scheme. We say that A is Byzantine-resilient if the sequence $\boldsymbol{\nabla} Q(\boldsymbol{x}_t) = 0$ converges almost surely to zero, despite the presence of up to f Byzantine workers.*

3. Kardam

In this section, we present the two main components of our algorithm, *Kardam*[2], namely the filtering and the dampening components. We also establish the theoretical guarantees of each component. For space limitations, the full corresponding proofs are given in the supplementary material.

3.1. Byzantine-resilient Filtering Component

The parameter server accepts a gradient $\boldsymbol{g}_p$ from worker p (i.e., updates the model with $\boldsymbol{g}_p$) if $\boldsymbol{g}_p$ is accepted by the Byzantine-resilient filtering component of Kardam. This component itself consists of a *Lipschitz filter* followed by a *frequency filter* that we describe in the following.

Lipschitz filter. This filter can be viewed as a kinetic validation at the parameter server based on the empirical Lipschitzness.

Definition 3 (Empirical Lipschitz coefficient)**.** *The empirical Lipschitz coefficient at worker p is defined as $\hat{K}_p = \frac{\|\boldsymbol{g}_p - \boldsymbol{g}_p^{prev}\|}{\|\boldsymbol{x}_{l_p} - \boldsymbol{x}_{l_p}^{prev}\|}$. The empirical Lipschitz coefficient at the parameter server is defined with respect to a received gradient from worker p and an updated gradient from worker q at the previous epoch ($t-1$) as $\hat{K}_t^p = \frac{\|\boldsymbol{g}_p - \boldsymbol{g}_q\|}{\|\boldsymbol{x}_t - \boldsymbol{x}_{t-1}\|}$.*

The empirical Lipschitz coefficient ($\hat{K}_p$) reflects the local empirical observation of the gradient evolution, normalized

[2]Kardam was a Bulgarian khan who pre-empted the Byzantine empire's invasion. He was the predecessor of *Krum* (Blanchard et al., 2017), the Bulgarian khan gave his name to the first provable solution for the synchronous Byzantine SGD problem.

by the model evolution. Each worker p derives this coefficient between the current and the previous models used to compute the current and previous gradients of p respectively.

The Lipschitz filter accepts the candidate gradient $\boldsymbol{g}_p$ if the empirical Lipschitzness for $\boldsymbol{g}_p$ (Definition 3) is not suspicious, i.e., if it is smaller than a median empirical Lipschitzness of all the workers as follows.

$$\hat{K}_t^p \leq \hat{K}_t \triangleq quantile_{\frac{n-f}{n}}\{\hat{K}_p\}_{p\in P}$$

where $quantile_{\frac{n-f}{n}}$ represents the element that separates the $\frac{n-f}{n}$ fraction of workers with the smallest empirical Lipschitz values from the remaining $\frac{f}{n}$ fraction with the highest values (i.e., the $(100 \cdot \frac{n-f}{n})^{th}$ percentile). We highlight that there exist two honest workers p_1 and p_2 such that $\hat{K}_{p_1} \leq \hat{K}_t \leq \hat{K}_{p_2}$ since our single dimensional median is guaranteed to be bounded by values from any group of size $n-f$ (i.e., group of honest workers).

The complexity of the Lipschitz filter is $\mathcal{O}(d+n)$ (computing distances on 2 d-dimensional vectors, then getting the median of n scalars, in $\mathcal{O}(n)$ with quick-select).

Obviously, the Lipschitz filter will end up filtering fast workers (that reach the more curved regions of the cost functions before the others) or slow workers (that are delayed in a curved region while everyone else is already in a less curved region). We note that this filter, roughly speaking, suspects f workers to be Byzantine and thus a pessimistic choice for f would increase the overhead of Kardam (filters more gradients due to a pessimistic choice for f).

Theorem 1 (Optimal Slowdown)**.** *We define the slowdown SL as the ratio between the number of updates from honest workers that pass the Lipschitz filter and the total number of updates delivered at the parameter server. We derive the upper and lower bounds of SL in the following.*

$$\frac{n-2f}{n-f} \leq SL \leq \frac{n-f}{n}$$

The lower and upper bounds are tight and hold when there are f Byzantine workers and no Byzantine workers respectively. Therefore Kardam achieves the optimal bounds with respect to any Byzantine-resilient SGD scheme and $n \approx 3f$ workers.

Proof. Any Byzantine-resilient SGD scheme assuming f Byzantine workers will at most use $\frac{n-f}{n}$ of the total available workers (upper bound). By definition, the Lipschitz filter accepts the gradients computed by $\frac{n-f}{n}$ of the total workers with empirical Lipschitzness below $\hat{K}_t$. If every worker is honest, then the filter accepts gradients from $\frac{n-f}{n}$ of the workers. We thus get the tightness of the upper bound for the slowdown of Kardam. For the lower bound, the Byzantine workers can know that putting a gradient proposition above $\hat{K}_t$ will get them filtered out and the parameter server will end up using only the honest workers available. The optimal attack would therefore be to slow down the server by getting tiny-Lipschitz gradients accepted while preventing the model from actually changing. This way, the Byzantine workers will make the server filter gradients from a total of f out of the $n-f$ honest workers, leaving only $n-2f$ useful workers for the server. □

Theorem 2 (Byzantine resilience in asynchrony)**.** *Let A be any distributed SGD scheme. If the maximum successive gradients that A accepts from a single worker and the maximum delay are both unbounded, then A cannot be Byzantine-resilient when $f \geq 1$.*

Proof. Without any restrictions, the parameter server could only accept successive gradients from the same Byzantine worker (without getting any update from any honest worker), for example, if the Byzantine worker is faster than any other worker (which is true by the definition of a Byzantine worker and by the fact that delays on (honest) workers are unbounded). This way, the Byzantine worker can force the parameter server to follow arbitrarily bad directions and never converge. Hence, without any restriction on the number of gradients from the workers, we prove the impossibility of asynchronous Byzantine resilience. Readers familiar with distributed computing literature might note that if asynchrony was possible for Byzantine SGD without restricting the number of successive gradients from a single worker, this could be used as an abstraction to solve asynchronous Byzantine consensus (that is impossible to solve (Fischer et al., 1985)). This provides another proof (by contradiction) for our theorem. □

Given Theorem 2 and the objective of making Kardam Byzantine-resilient in an asynchronous environment (i.e., while letting workers be arbitrarily delayed), we introduce the frequency filter.

Frequency filter. The goal of this filter is to limit the number of successive gradients [3] from a single worker to a value of f, thus not allowing the Byzantine workers to prevent honest workers from updating the model. Consider L as the list of workers who computed the last $2f$ accepted gradients. Assume that the candidate gradient $\boldsymbol{g}_q$ passes the Lipschitz filter. The frequency filter adds worker q at the end of L (i.e., at position $L[2f+1] = q$). If adding this candidate gradient $\boldsymbol{g}_q$ makes any set of workers of size f appear more than f times in L, then q is rejected, otherwise, q is accepted. For each worker p, the number of times p appeared in L is

[3] One open problem left in our work is the extent to which this filter is too harsh for asynchronous schemes. For instance, it can at least lead to a randomly shuffled round-robin schedule.

denoted by n_p. The frequency filter accepts a gradient from worker p if the following holds: $\sum_{p\in\theta} n_p \leq f$, where θ denotes the set of f workers with the f maxima of $\{n_p\}_{p=1}^n$. The time complexity of the frequency filter is $\mathcal{O}(2f+1)$ to compute $\{n_p\}_{i=1}^n$ (going through the list L of size $2f+1$), in addition to $\mathcal{O}(fn)$ to find the f maxima among $\{n_p\}_{i=1}^n$.

Lemma 1 (Limit of successive gradients)**.** *The frequency filter ensures that any sequence of length $2f+1$ consequently accepted gradients contains at least $f+1$ gradients computed by honest workers.*

Proof. Given any sequence of $2f+1$ consequently accepted gradients (L), we denote by S the set of workers that computed these gradients. The frequency filter guarantees that any f workers in S computed at most f gradients in L. At most f workers in S can be Byzantine, thus at least $f+1$ gradients in L are from honest workers. □

Given asynchrony (unbounded delays), we do not assume any upper bound on the norm of the model, the norm of the gradients or the values of the cost function (regularization schemes can make the loss grow arbitrarily and thereby the gradients norms). However, we assume (as in (Bottou, 1998)) that the cost function is lower bounded by a positive scalar. This assumption holds for all the standard cost functions that are at least lower bounded by zero (e.g., square loss, cross-entropy or any norm-based cost). We denote ***Kar*** by the sequence of gradients accepted (i.e., not filtered) by Kardam, and by $\boldsymbol{Kar}_t$ the gradient accepted by Kardam in epoch t.

Theorem 3 (Correct cone and bounded statistical moments)**.** *If $N > 3f+1$ then for any $t \geq t_r$ (we show that $t_r \in \mathcal{O}(\frac{1}{K\sqrt{|\xi|}})$ where $|\xi|$ is the batch-size of honest workers):*

$$\mathbb{E}(\|\boldsymbol{Kar}_t\|^r) \leq A'_r + B'_r\|\boldsymbol{x}_t\|^r \text{ for any r=2, 3, 4 and}$$

$$\langle \mathbb{E}(\boldsymbol{Kar}_t), \nabla Q_t\rangle = \Omega(1 - \frac{\sqrt{d}\sigma}{\|\nabla Q(\boldsymbol{x}_t)\|})\|\nabla Q(\boldsymbol{x}_t)\|^2.$$

Expectations are on any randomness up to time t conditioned on the past.

Proof. (Sketch) The frequency filter guarantees that there is always an update from a honest worker in any sequence of $f+1$ updates (Lemma 1), i.e., at any time t, there is an interval $t-i$ where $i < f+1$ such that the vector that passed the Lipschitz filter is a vector sent by an honest worker (therefore an unbiased estimation of the true gradient). With this in mind, and using triangle inequalities over a series of (at most f) previous updates, we prove inequalities on the r-th statistical moments of ***Kar***. Those inequalities are in turn plugged into the requirements for the (almost sure) global confinement argument of (Bottou, 1998).

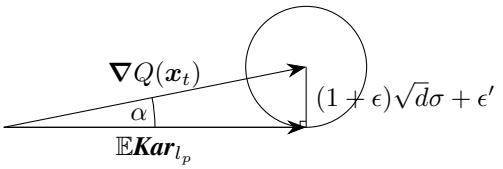


Figure 2: If $\left\|\mathbb{E}\boldsymbol{Kar}_{l_p} - \nabla Q(\boldsymbol{x}_t)\right\| \leq (1+\epsilon)\sqrt{d}\sigma + \epsilon'$ then $\langle \mathbb{E}\boldsymbol{Kar}_{l_p}, \nabla Q(\boldsymbol{x}_t)\rangle$ is upper bounded by $(1-\sin\alpha)\|\nabla Q(\boldsymbol{x}_t)\|^2$ where $\sin\alpha = \frac{(1+\epsilon)\sqrt{d}\sigma+\epsilon'}{\|\nabla Q(\boldsymbol{x}_t)\|}$.

With the guarantees of almost sure global confinement, and using the Liptschitz properties, and (again) the existence of honest (unbiased) workers in the "recent past" as explained above, we find the lower-bound of the scalar product between the two desired vectors $\langle \mathbb{E}(\boldsymbol{Kar}_t), \nabla Q_t\rangle$ when their distance is small enough compared to their own norms (Figure 2). This finally shows that Kardam remains in a cone of an angle α that is upper bounded by $\arcsin(\frac{(1+\epsilon)\sqrt{d}\sigma+\epsilon'}{\|\nabla Q(\boldsymbol{x}_t)\|})$ with appropriately chosen ϵ and ϵ'. □

3.2. Staleness-aware Dampening Component

We now present the component of Kardam that enables staleness-aware asynchronous updates for the ML model. For the sake of clarity, we denote the time by $t' \triangleq t - tr$ (Theorem 3). We introduce the *M-soft-async* protocol where the server updates the model only after receiving M gradients. The update rule for Kardam is the following.

$$\begin{aligned}\boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \gamma_t \boldsymbol{Kar}_t \\ &= \boldsymbol{x}_t - \gamma_t \sum_{[\boldsymbol{G}(\boldsymbol{x}_l;\xi_m),l]\in\mathcal{G}_t} \Lambda(\tau_{tl})\cdot \boldsymbol{G}(\boldsymbol{x}_l;\xi_m) \quad (1)\end{aligned}$$

where $\boldsymbol{G}(\boldsymbol{x}_l;\xi_m)$ denotes the gradient w.r.t the model parameters $\boldsymbol{x}_l$ on the mini-batch ξ_m. We assume that every gradient passes the filtering scheme (Section 3.1) at the epoch t. Kardam requires $|\mathcal{G}_t| = M$ gradients for each update.

The difference between the standard SGD update rule and our Equation 1 illustrates how Kardam handles asynchronous updates. Kardam dampens each update depending on its staleness value (τ_{tl}). Kardam employs a decay function $\Lambda(\tau_{tl})$ such that $0 \leq \Lambda \leq 1$ to derive the dampening factor for each distinct value of staleness.

Definition 4 (Dampening function)**.** *We employ a bijective and strictly decreasing dampening function $\tau \mapsto \Lambda(\tau)$ with $\Lambda(0) = 1$.*[4] *Note that every bijective function is also invertible, i.e., $\Lambda^{-1}(\nu)$ exists for every ν in the range of the Λ function.*

Let $\boldsymbol{\Lambda}_t$ be the set of Λ values associated with the gradients

[4] If $\Lambda(0) = 1$, then there is no decay for gradients computed on the latest version of the model, i.e., $\tau_{tl} = 0$.

at timestamp t.

$$\boldsymbol{\Lambda}_t = \{\Lambda(\tau_{tl}) \mid [g, l] \in \mathcal{G}_t\}$$

We partition the set $\mathcal{G}_t$ of gradients at timestamp t according to their Λ-value as follows.

$$\mathcal{G}_t = \bigsqcup_{\lambda \in \boldsymbol{\Lambda}_t} \mathcal{G}_{t\lambda}$$
$$\mathcal{G}_{t\lambda} = \{[g, l] \in \mathcal{G}_t \mid \Lambda(\tau_{tl}) = \lambda\}$$

Therefore, the update equation can be reformulated as follows.

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma_t \sum_{\lambda \in \boldsymbol{\Lambda}_t} \lambda \cdot \sum_{[\boldsymbol{G}(\boldsymbol{x}_l;\xi), l] \in \mathcal{G}_{t\lambda}} \boldsymbol{G}(\boldsymbol{x}_l; \xi)$$

Definition 5 (Adaptive learning rate)**.** *Given the Lipschitz constant K, the total number of timestamps T, and the total number of gradients in each timestamp as M, we define γ_t as follows.*

$$\gamma_t = \underbrace{\sqrt{\frac{Q(\boldsymbol{x}_1) - Q(\boldsymbol{x}^*)}{KTMd\sigma^2}}}_{\gamma} \cdot \underbrace{\frac{M}{\sum_{\lambda \in \boldsymbol{\Lambda}_t} \lambda |\mathcal{G}_{t\lambda}|}}_{\mu_t} \tag{2}$$

where γ is the baseline component of the learning rate and μ_t is the adaptive component that depends on the amount of stale updates that the server receives at timestamp t. Moreover, μ_t incorporates the total staleness at any timestamp t based on the staleness coefficients (λ) associated with all the gradients received in timestamp t. $Q(\boldsymbol{x}^*)$ (loss value at the optimum) can be assumed to be equal to zero, c.f. supplementary material (Definition 4) for additional comments.

Remark 1 (Correct cone)**.** *As a consequence of passing the filter and of Theorem 3, $\boldsymbol{G}$ satisfies the following.*

$$\langle \mathbb{E}_\xi \boldsymbol{G}(\boldsymbol{x}; \xi), \boldsymbol{\nabla} Q(\boldsymbol{x}) \rangle > \Omega((\|\boldsymbol{\nabla} Q(x_t)\| - \sqrt{d}\sigma)\|\boldsymbol{\nabla} Q(x_t)\|)$$

The theoretical guarantee for the convergence rate of Kardam depends on Assumptions 2,4 and Remark 1. These assumptions are weaker than the assumptions for the convergence guarantees in (Zhang et al., 2016b; Jiang et al., 2017). In particular, due to unbounded delays and the potential presence of Byzantine workers, we only assume the unbiased gradient estimator $\boldsymbol{G}(\cdot)$ for honest workers (Assumption 1). We instead employ (Remark 1) the fact that $\boldsymbol{G}(\cdot)$ and $\boldsymbol{\nabla} Q(\boldsymbol{x})$ make a lower bounded angle together (and subsequently a lower bounded scalar product) for all the workers. The classical unbiased assumption is more restrictive as it requires this angle to be exactly equal to 0, and the scalar product to be equal to $\|\boldsymbol{\nabla} Q(\boldsymbol{x})\| \cdot \|\boldsymbol{G}(\boldsymbol{x})\|$. Most importantly, we highlight the fact that those assumptions are satisfied by Kardam, since every gradient used in this section to compute the ***Kar*** update has passed the Lipschitz filter of the previous section.

Theorem 4 (Convergence guarantee)**.** *We express the convergence guarantee in terms of the ergodic convergence, i.e., the weighted average of the $\mathcal{L}_2$ norm of all gradients ($||\nabla Q(\boldsymbol{x}_t)||^2$). Using the above-mentioned assumptions, and the maximum adaptive rate $\mu_{\max} = \max\{\mu_1, \ldots, \mu_t\}$, we get the following bound on the convergence rate.*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\|\nabla Q(\boldsymbol{x}_t)\|^2 \leq (2 + \mu_{\max} + \gamma K M \chi \mu_{\max})\, \gamma K d\sigma^2 + d\sigma^2 + 2DK\sigma\sqrt{d} + K^2 D^2$$

under the prerequisite that

$$\sum_{\lambda \in \boldsymbol{\Lambda}_t} \lambda^2 |\boldsymbol{\Lambda}_t| \Bigg\{ K\gamma_t^2 + \sum_{s=1}^{\infty} \Big(\sum_{\nu \in \boldsymbol{\Lambda}_{t+s}} \gamma_{t+s} K^2 \nu |\mathcal{G}_{t+s,\nu}| \Lambda^{-1}(\nu) \mathbb{I}_{(s \leq \Lambda^{-1}(\nu))} \gamma_t^2 \Big) \Bigg\} \leq \sum_{\lambda \in \boldsymbol{\Lambda}_t} \frac{\gamma_t \lambda}{|\mathcal{G}_{t\lambda}|} \tag{3}$$

where the Iverson indicator function is defined as follows.

$$\mathbb{I}_{(s \leq \Delta)} = \begin{cases} 1 & \text{if } s \leq \Delta \\ 0 & \text{otherwise.} \end{cases}$$

It is important to note that the prerequisite (Inequality 3) holds for any decay function Λ (since $\lambda < 1$ holds by definition) and for any standard learning rate schedule such that $\gamma_t < 1$. Various SGD approaches (Zhang et al., 2016b; Lian et al., 2015; Zhang et al., 2016a; Jiang et al., 2017) provide convergence guarantees with similar prerequisites.

Theorem 5 (Convergence time complexity)**.** *Given any mini-batch size $|\xi|$, the number of gradients M the server waits for before updating the model, and the total number of epochs T, the time complexity for the convergence of Kardam is:*

$$\mathcal{O}\left(\frac{\mu_{\max}}{\sqrt{T|\xi|M}} + \frac{\chi \mu_{\max}}{T} + d\sigma^2 + 2DK\sigma\sqrt{d} + K^2 D^2\right)$$

where χ denotes a constant such that for all τ_{tl}, the following inequality holds:

$$\tau_{tl} \cdot \Lambda(\tau_{tl}) \leq \chi \tag{4}$$

Theorem 5 highlights the relation between the staleness and the convergence time complexity. This time complexity is linearly dependent on the decay bound (χ) and the maximum adaptive rate (μ_{max}).

Remark 2 (Dampening comparison)**.** *Given two dampening functions $\Lambda_1(\tau) = \frac{1}{1+\tau}$ and $\Lambda_2(\tau) = exp(-\alpha \sqrt[\beta]{\tau})$, and the convergence time complexity from Theorem 5, $\Lambda_2(\tau)$ converges faster than $\Lambda_1(\tau)$ when $\frac{\beta}{e} < \alpha \leq \frac{ln(\tau+1)}{\sqrt[\beta]{\tau}}$.*

We also empirically highlight Remark 2 by comparing these two functions in Figure 4 where DynSGD (Jiang et al., 2017) employs Λ_1 and Kardam employs Λ_2.

4. Experiments

In this section, we report on our empirical evaluation of our distributed implementation of Kardam. Experiments on Byzantine attacks are mostly illustrative for (1) the importance of the dampening component and (2) the overhead of the filtering component. Due to the intractability of testing all possible attacks, the only option is to prove Byzantine resilience mathematically and focus in the empirical part on the performance overhead of Kardam.

We employ the convolutional neural network (CNN) described in Table 2 for image classification on CIFAR-100. The chosen base learning rate is $15 * 10^{-4}$ and the mini-batch size is 100 examples (Neyshabur et al., 2015). If not stated otherwise, we employ a setup with no actual Byzantine behavior and deploy Kardam with $f = 3$ on 10 workers.

Parameters	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
Kernel size	32×32×3	3×3×16	3×3	3×3×64	4×4	384	192	100
Strides		1×1	3×3	1×1	4×4			

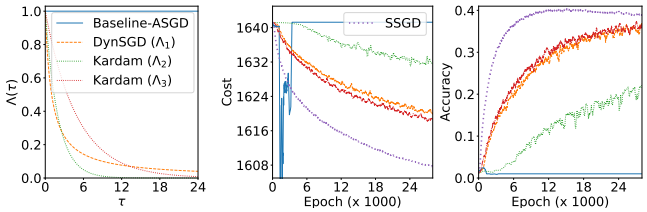
Table 2: CNN parameters for CIFAR-100.

Staleness-aware learning. We simulate a Gaussian staleness distribution (Zhang et al., 2016b) and evaluate Kardam with different dampening functions $\Lambda(\tau)$ (Definition 4) shown in Figure 3(a). We compare with the performance of Kardam without the Byzantine resilience component (BASELINE-ASGD) by using the constant function ($\Lambda_1 = 1$). Additionally, we compare with a state-of-the-art staleness-aware learning algorithm (DYNSGD (Jiang et al., 2017)) that employs an inverse dampening function ($\Lambda_2(\tau) = \frac{1}{1+\tau}$). Finally, we use two exponential functions ($\Lambda = exp(-\alpha * \tau)$) which, to the best of our knowledge, only Kardam enables.

Figure 3 depicts the very fact that the staleness-aware component of Kardam is crucial in asynchronous environments. We simulate a Gaussian distribution (Figure 3(b)) (similar to (Zhang et al., 2016b)) and show that SSGD has the faster convergence whereas BASELINE-ASGD diverges (Figures 3(b) and 3(c)).

Figure 3 also highlights the need for an adjustable smoothness on the dampening function. A very steep function (Λ_2) almost ignores many of the updates (weighted by a very small value) and thus suffers a slower convergence. A tuned exponential function (Λ_3) accelerates the convergence in comparison with the inverse function of DYNSGD. Moreover, Kardam (Λ_3) assigns larger weights to the less stale updates ($\tau < 13$) compared to DYNSGD and vice versa.

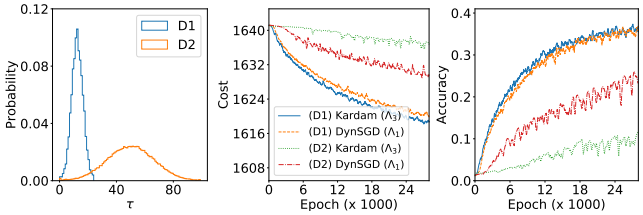
The dampening function selection is the outcome of adjusting the trade-off between the robustness and the magnitude of each update. We observe similar results for the EMNIST dataset (in our supplementary material) and thus highlight that the dampening function can be selected based on the expected staleness distribution, and not necessarily adjusted for each different application.



(a) Dampening functions (b) Cost (c) Accuracy

Figure 3: Staleness-aware learning for CIFAR-100. BASELINE-ASGD denotes Kardam without the dampening component and SSGD the ideal (synchronous) SGD execution. The staleness follows a Gaussian distribution ($mean = 12$, $\sigma = 4$) and the dampening functions are $\Lambda_1 = \frac{1}{\tau+1}$, $\Lambda_2 = e^{0.5\tau}$, $\Lambda_3 = e^{0.2\tau}$.

Impact of staleness. An increase in the amount of staleness leads to a slower convergence according to Theorem 5 (i.e., larger χ in Equation 4). Figure 4 depicts the impact of the amount of staleness on Kardam and DYNSGD for two different staleness distributions (D_1 and D_2). We observe that the smaller the mean of the distribution, the faster the convergence. We verify that Kardam outperforms DYNSGD for D_1 and vice versa for D_2, as the values of the exponential dampening function become very small for the larger staleness values (D_2). We highlight that our experimental setup includes significantly higher staleness (D2) than the competitors (Zhang et al., 2016b; Jiang et al., 2017).



(a) Staleness distributions (b) Cost (c) Accuracy

Figure 4: Impact of staleness for CIFAR-100.

Byzantine resilience. We observe that the overhead of the Byzantine resilience in the setup with no actual Byzantine behavior is only in terms of filtered (i.e., wasted) gradients and not in terms of convergence speed (in terms of epochs). Moreover, the drop ratio under the staleness distribution D_1 is 27.9% and 19.6% for Kardam employing Λ_1 and Λ_3 respectively, thus aligned with our theoretical bound (Theorem 1). The slowdown would decrease accordingly by decreasing f, i.e., being more optimistic about the number of Byzantine workers.

We test Kardam against a *baseline* Byzantine behavior (3 out of 10 workers send $g_p^{byz} = -10g_p$) and observe that Kardam successfully filters 100% of the Byzantine gradients (an empirical confirmation of the theoretically proven Byzantine resilience of Kardam).

5. Related Work and Concluding Remarks

Kardam is, to the best of our knowledge, the first asynchronous distributed SGD algorithm that tolerates Byzantine behavior. In the following, we discuss papers that either address asynchrony or Byzantine behavior.

Asynchronous stochastic gradient descent. SGD is used widely in ML solutions due to its convergence guarantees with low time complexity per update, low memory cost, and robustness against noisy gradients. Several variants of SGD have been proposed to improve the convergence rate and the robustness against noise. Stale-synchronous parallel (Ho et al., 2013; Cui et al., 2014) or bulk-synchronous (Zinkevich et al., 2010; Chen et al., 2016) variants typically target settings with limited staleness due to the limited performance variability among the computing devices. Other approaches consider variance minimization by importance sampling (Alain et al., 2015). The theoretical guarantees underlying these approaches assume synchronous updates as well as a specific formula to compute a gradient norm on each sample, which is only valid for multilayer perceptrons. The scheduler in (Zhang et al., 2016a) assumes all workers to be constantly available, which makes the algorithm not applicable to our setting with Byzantine workers. (Jiang et al., 2017) recently introduced a stale-synchronous parallel (SSP) heterogeneity-aware algorithm. SSP algorithms assume bounded staleness while Kardam guarantees convergence without any such bound (i.e., asynchronous parallel). Additionally, Kardam provides the flexibility of choosing the appropriate dampening function according to the expected staleness distribution while being Byzantine tolerant and asynchronous. We show both theoretically (Remark 2) and empirically (Figure 3) that an exponential dampening function leads to a faster convergence. (Kaoudi et al., 2017) recently proposed an elegant optimizer to predict the optimal SGD variant based on the expected cost per iteration and the estimated number of iterations. This estimation does not however account for stale updates. Our convergence analysis for Kardam could be employed to estimate the number of iterations for different dampening functions and hence to predict the optimal staleness-aware SGD variant. (Lock-freedom) Kardam, put *before* lock-free solutions such as Hogwild (Recht et al., 2011) would not break the convergence requirements (since the purpose of Kardam is to preserve them despite Byzantine workers). However, Kardam and Hogwild do not commute.

Second order methods. These methods rely on computing the Hessian matrix instead of the Lipschitz factor (Kardam filtering component). They were not specifically designed for Byzantine resilience but can in fact be employed for that purpose. However, unlike our scalar-based Lipschitz filter ($\mathcal{O}(d)$ time complexity that is already within the usual cost of an SGD update), they suffer from the curse of dimensionality. Moreover, the parameter server does no less than $\Omega(d^2)$ verifications on the Hessian matrix or on the gradient covariance matrix. In the presence of a cheap (constant size K) heuristic, the parameter server will let the Byzantine worker with a margin of $d^2 - K$ open coordinates to use for an attack. Since $d \gg K$ the heuristic alternative clearly hampers Byzantine resilience.

The differentiability lenses of Lipschitz. A central piece of our work is to filter out suspected vectors based on their (lack of) similar Lipschitzness with the median behavior. We prove that this *filtering* idea is sound, given that a significant fraction ($\Omega(\frac{n-f}{n})$) of workers will almost surely pass it and that Byzantine workers passing it are not harmful. In fact, leveraging the Lipschitzness properties, in the differentiable context of gradient-based learning, is not an uncommon idea. It was used in different contexts, for example, to understand fine-grained robustness, i.e robustness of the model to internal errors at the level of neurons and/or weights, this was done in (El Mhamdi & Guerraoui, 2016; 2017; El Mhamdi et al., 2017) proving a tight upper bound on the Lipschitz coefficient of neural networks, and deriving an exponential dependency with the depth and a polynomial dependency with the Lipschitz coefficient of the activation function used in each layer. In the same time, Lipschitzness was leveraged to compute spectral bounds as in (Cisse et al., 2017; Bartlett et al., 2017) both of which observed the same exponential dependency on the depth. In fact, manipulating differentiable objects is what makes the world of learning fundamentally different from the usual world of distributed computing, where the focus is on combinatorial and discrete structures. The differentiability of learning algorithms acts as a source of relaxation to solve a distributed computing task (estimating a gradient, distributively) in asynchrony and in the presence of Byzantine workers. The shorter the time it takes for Kardam to self-stabilize (t_r) the better in term of the speed of convergence. As we prove in Theorem 3, t_r is shorter with a larger global Lipschitz coefficient, i.e., steeper cost functions. Nevertheless, the cost function cannot be controlled. Yet, t_r can be decreased by increasing the batch size per worker, which is no surprise in learning theory (increasing the batch size is one of the most unavoidable taxes (Blanchard et al., 2017; Bousquet & Bottou, 2008; El Mhamdi et al., 2018) for increasing robustness). In practice, our experiments show no significant impact from t_r in the absence of actual Byzantine workers. In their presence, Kardam remains, to the best of our knowledge, the first provably Byzantine-resilient option to run SGD asynchronously.

An open problem now is how to tackle the Byzantine question in asynchronous machine learning *beyond* gradient-based algorithms. We argue that the core idea we present –filtering on quantiles from the recent past– could have applications to any approach where updates arrive with suspicions on either staleness or malicious behavior.

References

Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

Alain, Guillaume, Lamb, Alex, Sankar, Chinnadhurai, Courville, Aaron, and Bengio, Yoshua. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

Bartlett, Peter L, Foster, Dylan J, and Telgarsky, Matus J. Spectrally-normalized margin bounds for neural networks. In *NIPS*, pp. 6241–6250, 2017.

Biggio, Battista and Laskov, Pavel. Poisoning attacks against support vector machines. In *ICML*, 2012.

Blanchard, Peva, El Mhamdi, El Mahdi, Guerraoui, Rachid, and Stainer, Julien. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, pp. 118–128, 2017.

Bottou, Léon. Online learning and stochastic approximations. *Online learning in neural networks*, 17(9):142, 1998.

Bousquet, Olivier and Bottou, Léon. The tradeoffs of large scale learning. In *NIPS*, pp. 161–168, 2008.

Cachin, Christian, Guerraoui, Rachid, and Rodrigues, Luis. *Introduction to reliable and secure distributed programming*. 2011.

Chen, Jianmin, Monga, Rajat, Bengio, Samy, and Jozefowicz, Rafal. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Cisse, Moustapha, Bojanowski, Piotr, Grave, Edouard, Dauphin, Yann, and Usunier, Nicolas. Parseval networks: Improving robustness to adversarial examples. In *ICML*, pp. 854–863, 2017.

Cui, Henggang, Cipar, James, Ho, Qirong, Kim, Jin Kyu, Lee, Seunghak, Kumar, Abhimanu, Wei, Jinliang, Dai, Wei, Ganger, Gregory R, Gibbons, Phillip B, et al. Exploiting bounded staleness to speed up big data analytics. In *USENIX ATC*, pp. 37–48, 2014.

Cui, Henggang, Zhang, Hao, Ganger, Gregory R, Gibbons, Phillip B, and Xing, Eric P. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *European Conference on Computer Systems*, pp. 4, 2016.

Dean, Jeffrey, Corrado, Greg, Monga, Rajat, Chen, Kai, Devin, Matthieu, Mao, Mark, Senior, Andrew, Tucker, Paul, Yang, Ke, Le, Quoc V, et al. Large scale distributed deep networks. In *NIPS*, pp. 1223–1231, 2012.

El Mhamdi, E. M. and Guerraoui, R. When neurons fail. In *IPDPS*, pp. 1028–1037, May 2017.

El Mhamdi, E. M., Guerraoui, R., and Rouault, S. On the robustness of a neural network. In *SRDS*, pp. 84–93, Sept 2017.

El Mhamdi, El Mahdi and Guerraoui, Rachid. When neurons fail - technical report. pp. 19, 2016. Biological Distributed Algorithms Workshop, Chicago.

El Mhamdi, El Mahdi, Guerraoui, Rachid, and Rouault, Sébastien. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

Fischer, Michael J, Lynch, Nancy A, and Paterson, Michael S. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.

Ho, Qirong, Cipar, James, Cui, Henggang, Lee, Seunghak, Kim, Jin Kyu, Gibbons, Phillip B, Gibson, Garth A, Ganger, Greg, and Xing, Eric P. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, pp. 1223–1231, 2013.

Jiang, Jiawei, Cui, Bin, Zhang, Ce, and Yu, Lele. Heterogeneity-aware distributed parameter servers. In *SIGMOD*, pp. 463–478, 2017.

Kaoudi, Zoi, Quiané-Ruiz, Jorge-Arnulfo, Thirumuruganathan, Saravanan, Chawla, Sanjay, and Agrawal, Divy. A cost-based optimizer for gradient descent optimization. In *SIGMOD*, pp. 977–992, 2017.

Kurakin, Alexey, Goodfellow, Ian, and Bengio, Samy. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Lamport, Leslie, Shostak, Robert, and Pease, Marshall. The byzantine generals problem. *TOPLAS*, 4(3):382–401, 1982.

Li, Mu, Zhou, Li, Yang, Zichao, Li, Aaron, Xia, Fei, Andersen, David G, and Smola, Alexander. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, pp. 2, 2013.

Li, Mu, Andersen, David G, Park, Jun Woo, Smola, Alexander J, Ahmed, Amr, Josifovski, Vanja, Long, James, Shekita, Eugene J, and Su, Bor-Yiing. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, pp. 3, 2014a.

Li, Mu, Andersen, David G, Smola, Alexander J, and Yu, Kai. Communication efficient distributed machine learning with the parameter server. In *NIPS*, pp. 19–27, 2014b.

Lian, Xiangru, Huang, Yijun, Li, Yuncheng, and Liu, Ji. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, pp. 2737–2745, 2015.

Lian, Xiangru, Zhang, Huan, Hsieh, Cho-Jui, Huang, Yijun, and Liu, Ji. A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order. In *NIPS*, 2016.

Liu, Ji, Stephen, and Wright, J. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. Technical report, SIAM Journal on Optimization, 2015.

Lynch, Nancy A. *Distributed algorithms*. 1996.

Muñoz-González, Luis, Biggio, Battista, Demontis, Ambra, Paudice, Andrea, Wongrassamee, Vasin, Lupu, Emil C, and Roli, Fabio. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Workshop on Artificial Intelligence and Security*, pp. 27–38, 2017.

Neyshabur, Behnam, Salakhutdinov, Ruslan R, and Srebro, Nati. Path-sgd: Path-normalized optimization in deep neural networks. In *NIPS*, pp. 2422–2430, 2015.

Recht, Benjamin, Re, Christopher, Wright, Stephen, and Niu, Feng. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pp. 693–701, 2011.

Su, Lili. *Defending distributed systems against adversarial attacks: consensus, consensus-based learning, and statistical learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2017.

Zhang, Ruiliang, Zheng, Shuai, and Kwok, James T. Asynchronous distributed semi-stochastic gradient optimization. In *AAAI*, pp. 2323–2329, 2016a.

Zhang, Wei, Gupta, Suyog, Lian, Xiangru, and Liu, Ji. Staleness-aware async-sgd for distributed deep learning. In *IJCAI*, pp. 2350–2356, 2016b.

Zinkevich, Martin, Weimer, Markus, Li, Lihong, and Smola, Alex J. Parallelized stochastic gradient descent. In *NIPS*, pp. 2595–2603, 2010.