ByRD\textit{DiE}: Byzantine-resilient distributed coordinate descent for decentralized learning

Zhixiong Yang and Waheed U. Bajwa

\begin{abstract}
Distributed machine learning algorithms enable processing of datasets that are distributed over a network without gathering the data at a centralized location. While efficient distributed algorithms have been developed under the assumption of faultless networks, failures that can render these algorithms nonfunctional indeed happen in the real world. This paper focuses on the problem of Byzantine failures, which are the hardest to safeguard against in distributed algorithms. While Byzantine fault tolerance has a rich history, existing work does not translate into efficient and practical algorithms for high-dimensional distributed learning tasks. In this paper, two variants of an algorithm termed \textit{Byzantine-resilient distributed coordinate descent} (ByRD\textit{DiE}) are developed and analyzed that solve distributed learning problems in the presence of Byzantine failures. Theoretical analysis as well as numerical experiments presented in the paper highlight the usefulness of ByRD\textit{DiE} for high-dimensional distributed learning in the presence of Byzantine failures.
\end{abstract}

\textbf{Index Terms—}Byzantine failure, consensus, distributed optimization, empirical risk minimization, machine learning

\section{I. Introduction}

One of the fundamental goals in machine learning is to learn a model that minimizes the statistical risk. This is typically accomplished through the use of stochastic optimization techniques, with the underlying principle referred to as \textit{empirical risk minimization} (ERM) \cite{10,11,12}. The ERM principle involves the use of a training dataset and tools from optimization theory. Traditionally, the training data has been assumed available at a centralized location. Many recent applications of machine learning, however, involve the use of a dataset that is either distributed across different locations (e.g., the \textit{Internet of Things}) or that cannot be processed at a single machine due to its size (e.g., social network data). The ERM framework in this setting of distributed training data is often referred to as \textit{decentralized} or \textit{distributed} learning \cite{13,14}.

While excellent works have been done to solve the distributed learning problems, all these works make a simplified assumption that all nodes in the network function as expected. Practically speaking, the assumption does not always hold when unexpected interference appears; examples include cyber attacks, malfunctioning equipments and undetected failures \cite{15,16}. When a node arbitrarily deviates from its intended behavior, it is termed Byzantine failure \cite{17,18,19}. Byzantine failure is hard to detect in general but can easily jeopardize the whole network \cite{20,21,22}.

With just a simple strategy, one can show that one Byzantine node in the network will lead to failures of most state-of-the-art distributed learning algorithms \cite{23}. The main contribution of this paper is to introduce two variants of an algorithm that completes the distributed learning task in the presence of Byzantine failures.

\section{A. Related works}

To achieve the goal of distributed learning, one usually forms a distributed optimization problem by defining and minimizing, for each node, a loss function on its training set. Then the problem can be solved by applying distributed optimization algorithms. There are several types of distributed algorithms introduced to solve this problem. For example, there are gradient-based methods such as distributed gradient descent (DGD) \cite{24,25,26}. Gradient-based methods usually have lower local computational complexity. Another example is augmented Lagrangian-based methods \cite{27,28}, which iteratively update the primal variable and the dual variable. These methods require the ability to solve an optimization problem locally. Although the algorithms mentioned above can be used to solve a distributed learning problem, they all make an assumption that there is no failure in the network. Byzantine-resilient algorithms have been studied extensively over years \cite{29,30}. Byzantine-resilient algorithms for scalar case averaging consensus is introduced in \cite{31}. The algorithm proposed in \cite{32} extends the work from scalar consensus to scalar case optimization problems but the algorithm cannot be applied to vector settings. The work in \cite{33} introduces a method to implement distributed support vector machine (SVM) under Byzantine setting but the method does not generalize to general machine learning problems. A recent work in \cite{34} solves a vector setting distributed learning problem under Byzantine setting, yet the method requires a central processing center, which is not a fully-distributed solver so that the method does not work under fully-distributed settings.

\section{B. Our contribution}

There are limitations of existing works like \cite{32,33,34,35}. One of the limitations is that the algorithms proposed are pursuing the optimum of a convex combination of local empirical risk functions. Applications of such optimization algorithms, like machine learning, are usually modeled as a distributed optimization problem that pursues the minimum
of exact average of local empirical risk functions which is used as an approximation of the minimum of the true risk function. So far there are no guarantees that the outputs of these algorithms are close to either the empirical minimum or the true minimum. Another limitation is that when working with vectors, the existing Byzantine-resilient algorithms make strong assumptions on the network topology which depends on the dimensionality of a problem. These assumptions become impractical as the dimensionality of a problem increases.

Our work has two main contributions. First, we propose two algorithms that scale well with the dimensionality of distributed learning problems under the assumption that the training samples are independent and identically distributed (i.i.d), which is easily satisfied in the real world. Then we give theoretical guarantees that the output of the proposed algorithms will be close to the true minimum with high probability. Specifically, we show that the outputs of our algorithms converge to the ground truth faster than using only local information by a factor $\sum_2^3$ which will be defined in the next section.

C. Notations and organization

In this paper, we use $|·|$ to represent the size of a set. A vector $a$ is a column vector and $a^T$ represents the transpose of $a$. Index $[a]_k$ represents the $k$-th element of $a$. We use $||a||$ to denote the $\ell_2$-norm of $a$. For a matrix $A$, index $[A]_{ji}$ represents the element at the $j$-th row and $i$-th column. The indexes and iteration indicators $j, i, k, t, q, m, n$ are all integers. We use $I$ to denote the identity matrix and $1$ to denote a column vector whose elements all equal to 1. We use $\geq$ and $\leq$ to represent element-wise inequality. We use $\nabla f(w, (x, y))$ to denote the gradient of a function $f(w, (x, y))$ with respect to $w$.

The rest of this paper is organized as the following. Section II gives the problem formulation. Section III introduces our algorithm along with theoretical guarantees and respective proof. Numerical analysis is done in Section IV. Finally, section V concludes the paper.

II. PROBLEM FORMULATION

Given a network in which each node has access to some local training data, the main goal of this paper is to develop an algorithm to learn from these data in a distributed fashion when there is Byzantine failure in the network. The definition of Byzantine failure will be given later in the section. First let us describe the mathematical model.

A. Learning model

We consider a network of $M$ nodes, expressed as a directed, static graph $G(J, E)$. Here, the set $J := \{1, \ldots, M\}$ represents nodes in the network, while the set of edges $E$ represents communication links between different nodes. Specifically, $(j, i) \in E$ if and only if node $i$ can receive information from node $j$ and vice versa. Each node $j$ has access only to a local training set $S_j = \{(x_{jn}, y_{jn})\}_{n=1}^{S_j}$. Let $x \in \mathbb{R}^P$ represent the training features satisfying $\|x\| \leq B$ for some constant $B$ and $y$ be its label. For classification, $y \in \{-1, 1\}$, and for regression, $y \in \mathbb{R}$. We assume that all training samples are i.i.d. and drawn from an unknown distribution $D$, i.e., $(x_{jn}, y_{jn}) \sim D$. For simplicity, we assume that the size of local training sets are the same, i.e., $|S_j| = N$. The generalization to the case when local sizes are not equal is trivial. Ideally, we want to collect all the data into one set $S = \{(x_n, y_n)\}_{n=1}^{|S|}$ and perform centralized training on $S$. The goal of learning is to learn a function that correctly maps $x$ to $y$. One broadly used mapping function is $y = w^T x$. Sometimes this function is defined as $y = w^T x + b$, which can be transformed into $y = w^T x$ by adding one more dimension to $x$. To find a good $w$, first define a loss function $\ell(w, (x, y))$, where the value of loss function increases when the difference between mapping of $x$ and $y$ increases. To avoid over fitting, people usually add a regularizer to it. Then we can solve for $w$ by minimizing a regularized loss function $f(w, (x, y)) = R(w) + \ell(w, (x, y))$, where the regularizer $R(w)$ is assumed to be strongly convex and smooth e.g., $\frac{1}{2}\|w\|^2$. The regularized function is often referred to as risk function. In this paper, we focus on the class of convex differentiable loss functions; to name a few, square loss $(1 - yw^T x)^2$, square hinge loss $\max(0, 1 - yw^T x)^2$ and logistic loss $\ln(1 + e^{-yw^T x})$. We assume that the gradient of the loss function $\ell(w, (x, y))$ is $L$-Lipschitz continuous. Since $R(w)$ is smooth, we formally state this assumption as follows. 

**Assumption 1:** The risk function $f = R(w) + \ell(w, (x, y))$ satisfies $\|\nabla f(w_1, (x, y)) - \nabla f(w_2, (x, y))\| \leq L\|w_1 - w_2\|$.

When Assumption 1 is true, it is also true that the risk function itself is also Lipschitz, i.e., $\|f(w_1, (x, y)) - f(w_2, (x, y))\| \leq L\|w_1 - w_2\|$. We define the feasible set of $w$ as $W$. Then we can define the true mapping as $w^* = \arg\min_{w \in W} E_{(x, y) \sim D} f(w, (x, y))$. Since we do not know $D$, we cannot solve for $w^*$ directly. A broadly adopted alternative is to minimize the empirical risk $\hat{f}(w, S) = \frac{1}{|S|} \sum_{n=1}^{|S|} f(w, S) = R(w) + \frac{1}{|S|} \sum_{n=1}^{|S|} \ell(w, (x_n, y_n))$. The minimum of empirical loss can be shown to converge to $w^*$ with high probability as the sample size $|S|$ increases.

In some cases the training data cannot be made available at a single location. So we wish to learn the mapping in a distributed fashion, which can be done by employing distributed optimization algorithms. The main idea of distributed optimization is to minimize the average of all local empirical risk functions, i.e., $\sum_{j=1}^M \hat{f}(w, S_j) = R(w) + \frac{1}{MN} \sum_{j=1}^M \sum_{n=1}^N \ell(w, (x_{jn}, y_{jn}))$. To perform the distributed algorithm, we need nodes to cooperate with each other by communicating over edges. We define the neighbourhood of $j$ as $N_j := \{i \in J : (i, j) \in E\}$. Then we say that node $i$ is a neighbour of node $j$ if $i \in N_j$. Classic distributed algorithms proceed iteratively. In each iteration, a node is expected to accomplish two tasks: updating a local variable $w_j$ according to some function $g_j(\cdot)$ and broadcasting the local variables.
During the broadcasting step, a node \( i \) can receive values from node \( j \) only if \( j \in N_i \). The algorithm we propose in the next section will accomplish the following task: when a node \( j \) follows the algorithm, it can (i) achieve consensus, i.e., \( w_j = w_i \) \( \forall i \in J \) as the number of iterations \( t \to \infty \); (ii) learn a \( w_j \rightarrow w^* \) as sample size \( N \to \infty \).

**B. Byzantine failure model**

While distributed learning via message-passing has been well-understood \([19], [28], [29]\), such protocols require all nodes in the network to operate as intended. In contrast, the main assumption in this paper is that some of the nodes in the network can undergo Byzantine failure, which is formally defined below.

**Definition 1:** A node \( j \in J \) is said to be Byzantine if during any iteration, it updates its local variable using an update function \( g_j(\cdot) \neq g_j(\cdot) \) or broadcasts some value other than the intended update.

Note that different nodes can receive different values from the same Byzantine node in each message passing. Specifically, we assume there are at most \( b \) Byzantine nodes in the network. Let \( J' \) denote the set of non-faulty nodes. Without loss of generality, we assume that non-faulty nodes are labelled from 1 to \( |J'| \). Then we make some definitions and assumptions aligned with existing works e.g. \([13]\).

**Definition 2:** A subgraph \( G_r \) of graph \( G \) is called a reduced graph if it is generated from graph \( G \) by (i) removing all Byzantine nodes along with all their incoming and outgoing edges and (ii) removing additionally up to \( b \) incoming edges from each non-faulty node.

**Definition 3:** A source component of graph \( G_r \) is the collection of nodes in which each node in the source component has a directed path to every other node in \( G_r \).

**Assumption 2:** All reduced graphs \( G_r \), generated from \( G(J,E) \) contain a source component of size at least \( b + 1 \).

The assumption is to ensure that there is enough redundancy in the graph to tolerate Byzantine failures, which will help us later in the proof of theorems. It is obvious that the total number of different reduced graphs we can generate from \( G \) is finite as long as the number of nodes is finite. So the assumption can be checked for any graph with finite number of nodes. But how to check the assumption efficiently still remains an open topic. However, our experiments show that in a randomly connected graph, if the ratio between average incoming degree and number of Byzantine nodes \( b \) is high enough, the assumption is usually satisfied. Note that it is not necessary to know the exact \( b \). In application, we can set \( b \) as an upper bound of Byzantine nodes. As long as the network topology assumption is satisfied, the algorithm can tolerate up to \( b \) Byzantine nodes. Our goal then is to develop a Byzantine fault-tolerant algorithm for distributed learning under this main assumption.

---

**Algorithm 1 ByRD\( \text{iE with exact line search}\)**

**Input:** \( b, \epsilon, S_1, S_2, \ldots, S_M, \{\rho(t)\}^t_{t=1} \)

1. **initiate** \( w_j(0) = 0, j = 1, 2, \ldots, |J'|, t = 0 \)
2. **repeat**
   3. **for** \( k = 1, 2, 3, \ldots, P \)
      4. **repeat**
         5. \( t \leftarrow t + 1 \)
         6. **for** \( j = 1, 2, 3, \ldots, |J'| \) **do**
            7. Broadcast \( [w_j(t)]_k \) to all nodes in \( N_j \)
            8. Sort \( \{[w_i(t)]_k\} \) in increasing order \( \forall i \in N_j \)
            9. Define \( \mathcal{N}^g_j(t) = \arg \min \sum_{i \in X} [w_i(t)]_k \)
            10. Define \( \mathcal{N}^p_j(t) = \arg \max \sum_{i \in X} [w_i(t)]_k \)
            11. Define \( \mathcal{N}^{g,p}_j(t) = \mathcal{N}^g_j(t) \setminus \mathcal{N}^p_j(t) \)
            12. Update \( w_j \) as \( w_j(t+1)]_k = \frac{1}{|N_j|-2b+1} \sum_{i \in \mathcal{N}^{g,p}_j(t)} [w_i(t)]_k \)
            13. \( \rho(t) = \sqrt{f(w_j(t), S_j)} \) \( k \)
         14. **end for**
3. **end for**
15. **until** \( |[w_j(t)]_k - [w_i(t)]_k| < \epsilon, \forall i, j \in J' \)
20. **end for**
21. **until** Stop condition satisfied

**Output:** \( w_j, \forall j \in J' \)

---

**III. ByRD\( \text{iE: Byzantine-resilient distributed coordinate descent}\)**

As described in the previous section, we want to find the global optimal, i.e.,

\[
\text{Arg min}_w R(w) + \frac{1}{MN} \sum_{j=1}^{M} \sum_{n=1}^{N} l(w,(x_{jn}, y_{jn}))
\]

and expect that \( w_{\text{opt}} \to w^* \) as \( N \to \infty \). It is shown in \([30]\) that the exact global optimal of (1) is not achievable when \( b \geq 1 \). Existing works pursue a convex combination of local empirical risks as an alternative. In this section we introduce an algorithm called Byzantine-Resilient Distributed coordinate dEscent (ByRD\( \text{iE}) \) along with two different ways of implementing it. We will also show, for each implementation separately, that the algorithm pursues the true optimal in the presence of Byzantine failures given the training data is i.i.d..

**A. Implementing ByRD\( \text{iE with exact line search}\)**

As mentioned in section \([1]\) existing works have already introduce methods that solve scalar optimization problems under Byzantine setting. Another observation is that coordinate descent breaks vector optimization problems into a sequence of scalar optimizations. The idea of ByRD\( \text{iE} \) is indeed the combination of these two methods. One obvious way of this combination is employing algorithms in \([13]\) to perform exact line search at each dimension. The exact line search implementation we propose is shown in Algorithm \([13]\). This implementation can be broken into an outer loop (step 3)
and an inner loop (step [6]). The outer loop is the coordinate descent loop which breaks the vector optimization problem down to scalar problems. The inner loop solves a scalar-case optimization problem and ensures resilience to Byzantine failures. As we will explain later in this section, the inner loop will return the minimum of a convex combination of local empirical risks with respect to one dimension of $w$. We can show that the output of inner loop will converge to the true minimum of the scalar optimization problem almost surely (a.s.) as $N \to \infty$. Then we explain how we can embed the inner loop into the coordinate descent.

At the beginning of each inner loop, every node initiates from $w_j(0) = 0$ or some arbitrary value. Then during each iteration $t$, every node will complete the following: broadcasting, selection and updating. In the broadcasting step, node $j$ receives $[w_t(i)]_k$ for all neighbors $i \in N_j$. A node can receive values from both Byzantine and non-faulty neighbors. The main idea of selection is to remove some values that are “too large” or “too small” so that the values being used for update in each iteration will be upper and lower bounded by a set of values generated by non-faulty nodes. We break $N_j$ into 3 subsets $N_{j}^{b}$, $N_{j}^{f}$ and $N_{j}^{s}$, which are defined as the following:

$$N_{j}^{b}(t) = \arg\min_{X:|X|\le b} \sum_{i\in X} [w_t(i)]_k,$$  \hspace{1cm} (2)

$$N_{j}^{f}(t) = \arg\max_{X:|X|\le b} \sum_{i\in X} [w_t(i)]_k,$$  \hspace{1cm} (3)

$$N_{j}^{s}(t) = N_j \setminus N_{j}^{b}(t) \setminus N_{j}^{f}(t).$$  \hspace{1cm} (4)

One way of finding the three sets is by a sorting process. Node $j$ sorts $w_t$’s in an increasing order for all $i \in N_j$, breaking ties arbitrarily. Then node $j$ adds the neighbors that broadcast the largest $b$ $w_t$’s into $N_{j}^{b}$. Similarly we can define $N_{j}^{s}$. Then add all the neighbors that are not in $N_{j}^{b}$ or $N_{j}^{s}$ into $N_{j}^{f}$. The step is called selection because node $j$ only takes $w_t$’s from $N_{j}^{s}$ to update its local value. Note that there might still be $w_t$’s received from Byzantine nodes in $N_{j}^{s}$. The rule for update is described by the equation

$$[w_j(t+1)]_k = \frac{1}{|N_j| - 2b + 1} \sum_{i\in N_{j}^{f}(t), j} [w_t(i)]_k,$$  \hspace{1cm} (5)

where $\{\rho(t)\}_{t=1}^{\infty}$ is a sequence of stepsizes such that (s.t.) $0 < \rho(t+1) \le \rho(t), \sum_{t=1}^{\infty} \rho(t) = \infty$ and $\sum_{t=1}^{\infty} \rho^2(t) < \infty$. For inner loop to work properly, the graph needs to satisfy Assumption[2][13].

Inner loop is designed to solve a scalar case optimization problem. For simplicity, we define functions $h_k$ and $H_k$ only with respect to a scalar as

$$h_k(w', S) = f(w, S)[w]_k = w',$$  \hspace{1cm} (6)

and

$$H_k(w', S) = \sum_{j=1}^{M} \alpha_j \hat{f}(w, S_j)[w]_k = w'$$  \hspace{1cm} (7)

for some $\alpha_j \ge 0$ and $\sum_{j=1}^{M} \alpha_j = 1$. Then the minimum of the empirical risk with respect to one dimension is

$$\hat{w} = \arg\min_{w'} H(w', S)$$  \hspace{1cm} (8)

when initiating inner loop at some $w$. For the same $w$, define

$$w^{**} = \arg\min_{w'} \mathbb{E}_{(x,y) \sim D} h_k(w', S).$$  \hspace{1cm} (9)

**Lemma 1:** Let Assumption[1] and [2] hold. Initiate the inner loop at some $w$ then as $t \to \infty$, $[w_j(t)]_k \to \hat{w}$ for some $\alpha_j \ge 0$ and $\sum_{j=1}^{N} \alpha_j = 1$.[13]

The detailed proof for lemma 1 can be found in [13]. Lemma 1 shows that we can find the minimum of a convex combination of local empirical risk functions with respect to each dimension in the presence of Byzantine failure. Next, given that the training samples are i.i.d across all the local datasets, we can show that this minimum converges to the true minimum with high probability. Before showing the convergence, we need one more assumption.

**Assumption 3:** For any $w \in W$, the loss function $\ell(w, (x,y))$ has finite value over all training samples, i.e., $\ell(w, (x,y)) \le C < \infty, \forall (x,y) \in \bigcup_{j \in J} S_j$.

This assumption ensures that we are only operating with finite values. This assumption is usually satisfied by machine learning tasks in the real world because the variables and parameters in the real world are finite most of the time.

**Lemma 2:** Let Assumption[3] hold true. Given any $0 < \delta < 1$, with probability at least $1-\delta$,

$$|\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x,y)) - \mathbb{E}_{(x,y) \sim D} h(w^{**}, (x,y))| \le \frac{2C^2 \ln \frac{4}{\delta} \sum_{j=1}^{M} \alpha_j^2}{N}.$$  \hspace{1cm} (10)

The lemma shows that the minimum of a convex combination of the local empirical risk functions also converges to the minimum of true risk function as sample size $N$ increases. Note that $\frac{1}{M} \le \sum_{j=1}^{M} \alpha_j^2 \le 1$. So when $\alpha_j = \frac{1}{M}$ for all $j \in J$, i.e.,

$$|\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x,y)) - \mathbb{E}_{(x,y) \sim D} h(w^{**}, (x,y))| \le \frac{2C^2 \ln \frac{4}{\delta}}{MN},$$  \hspace{1cm} (11)

which is pursuing the exact average of local risks, we achieve the fastest rate of convergence in [10]. On the other extreme, when $M = 1$ and $\alpha_1 = 1$, we have

$$|\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x,y)) - \mathbb{E}_{(x,y) \sim D} h(w^{**}, (x,y))| \le \frac{2C^2 \ln \frac{4}{\delta}}{N}.$$  \hspace{1cm} (12)

Inequality (12) represents the convergence rate of using only $N$ local samples, which is the slowest rate we can have in [10]. This tells us that although we can no longer achieve the fastest convergence rate because of the Byzantine failure, the
convergence rate can still benefit from the cooperation among $M$ nodes by a factor $\sum_{j=1}^{M} \alpha_j^2$. The proof for lemma 2 is in Appendix A.

So far we have shown that we can solve a scalar optimization problem with the presence of Byzantine failure. Next we can employ this idea to perform coordinate descent. Lemma 1 shows that the inner loop can bring all $[w_j')]$‘s arbitrarily close to each other. So a consensus criteria $\epsilon$ should be set according to the requirements of different applications. The inner loop stops when the consensus condition is satisfied. The outer loop stops when some stop condition is satisfied (e.g., after certain number of rounds or the changes of $w_j$’s are small). Let us use $w(r)$ to denote the output of the $r$-th round of outer loop.

To better describe the process, we need to form a new vector $\bar{w}(r, k, w') = [(w(r))_1 \cdots (w(r))_{k-1} \ w' \ (w(r-1))_k \cdots (w(r-1))_1]^T$. (13)

The meaning of this vector $\bar{w}(r, k, w')$ is to let the first $k - 1$ dimensions equal to $w(r)$, which means that these dimensions have been updated at the $r$-th round. The $k$-th dimension of $\bar{w}(r, k, w')$ is equal to $w'$, which is to be optimized by the current round of inner loop. Then the dimensions with indexes greater than $k$ are to be determined in the future rounds of inner loops. In this way, we can express the output of the $r$-th outer loop and $k$-th inner loop as $\bar{w}(r, k, [w(r)]_k)$. With the help of this expression, we next show how to find $w(r)$. Define a vector $\beta_j(r) \in \mathbb{R}^P$ for each node $j$. Let $[\beta_j(r)]_k$ denote the weight for loss $\ell(\bar{w}(r, k, w'), (x_{jn}, y_{jn}))$ similar to the $\alpha_j$’s in the scalar case. So $\beta_j$’s satisfy $\beta_j \geq 0$ and $\sum_{j=1}^{M} [\beta_j]_k = 1$ for $1 \leq k \leq P$. Recall that the output of inner loop is the optimum of a scalar case optimization problem, i.e.,

$$[w(r)]_k = \arg\min_{w'} \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} \beta_j(r)_k \ell(\bar{w}(r, k, w'), (x_{jn}, y_{jn}))$$

$$= \arg\min_{w'} F_k(\bar{w}(r, k, w'), S).$$

Lemma 3: Let Assumption 1, 2 and 3 hold. If $w(r)$ is the output after $r$ rounds of outer loop, then when $N \to \infty$, $F_k(\bar{w}(r, k, [w(r)]_k), S) \to \min_{w'} \mathbb{E}_{(x,y)} \mathcal{D} f(\bar{w}(r, k, w'), (x, y))$ a.s. for all $r > 0$, $1 \leq k \leq P$ simultaneously.

The proof of Lemma 3 can be found in Appendix B. This Lemma will help us develop the next theorem.

Theorem 1: Let Assumption 1, 2 and 3 hold. Let $w(r)$ denote the output after $r$ rounds of Algorithm 1. Then as $N \to \infty$ and $r \to \infty$, $\mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y)) \to \mathbb{E}_{(x,y)} \mathcal{D} f(w^*, (x, y))$.

Note that both the empirical risk and the true risk functions are strongly convex. So the theorem can also be interpreted as $w(r) \to w^*$ due to the uniqueness of minimum of strongly convex functions. The development of this theorem is very similar to proving the convergence of a coordinate descent algorithm. By Lemma 3, we are given that at each dimension, we can find the a point that is closer to the minimum than the current point. Then we want to show that the algorithm will eventually converge to the minimum.

First we make two observations on $[13]$. (15)

$$[w(r, k+1, \ [w(r-1)]_{k+1}] = [w(r, k, [w(r)]_k)]$$

and

$$w(r) = \bar{w}(r, P, [w(r)]_P) = \bar{w}(r+1, 1, [w(r)]_1).$$

Then by Lemma 3 we know that as $N \to \infty$, we can always find a $[w(r)]_{k+1}$ satisfying

$$\mathbb{E}_{(x,y)} \mathcal{D} f(\bar{w}(r, k + 1, [w(r)]_{k+1}), (x, y)) \leq \mathbb{E}_{(x,y)} \mathcal{D} f(\bar{w}(r, k + 1, [w(r) - 1]_{k+1}), (x, y))$$

(17)

Then we compare the output of the round $r - 1$ and round $r$,

$$\mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y)) = \mathbb{E}_{(x,y)} \mathcal{D} f(\bar{w}(r, P, [w(r)]_P), (x, y))$$

(18)

Inequality (a) can be concluded by recursively applying (17) from $k = 1$ to $k = P - 1$. The inequality of (18) tells us that $\mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y))$ is non-increasing. Recall that $\mathbb{E}_{(x,y)} \mathcal{D} f(w^*, (x, y)) \leq \mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y)) \forall w$, which means that sequence $\mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y))$ converges as $r \to \infty$. Since $f(w(r), (x, y))$ is strongly convex, $\mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y)) = \mathbb{E}_{(x,y)} \mathcal{D} f(w(r+1), (x, y))$ if and only if $\nabla \mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y)) = 0$. So $\forall w(r) \neq w^*$, $\exists w(r+1)$ satisfying $\mathbb{E}_{(x,y)} \mathcal{D} f(w(r+1), (x, y)) < \mathbb{E}_{(x,y)} \mathcal{D} f(w(r), (x, y))$, which implies Theorem 1.

B. ByRDIE with switching coordinate

If we take a close look at ByRDIE with exact line search, we can find that each iteration of inner loop requires one round of communication and we need to run many rounds of inner loop for each dimension, which is inefficient in the sense of commincation. The reason is that we are trying to do exact line search. A more efficient way of implementing coordinate descent is to make one step along the direction of descent in one dimension and switch to another [32]. We propose another implementation of ByRDIE as shown in Algorithm 2 called switching coordinate. We still use the definition for $w$ in (13), where we use $t$ to denote each iteration instead of $r$.

In Algorithm 1, we fix $w_j$’s for all dimensions but dimension $k$ and update $[w_j]_k$ until all non-faulty nodes agree on the same $[w_j]_k$. However, in Algorithm 2 every dimension of $w_j$ could change before we reach consensus for any $k$. So Theorem 1 can not be applied to Algorithm 2. We need to show that, (a) non-faulty nodes agree on the same vector $V(t)$ as $t \to \infty$; (b) $V(t) \to w^*$ as $N \to \infty$.

Theorem 2: Let Assumption 1, 2 and 3 hold. Implementing ByRDIE with switching coordinate guarantees that $w_j(t) \to w^*$ as $t \to \infty$ when $N \to \infty$, $\forall j \in J'$. 

Let $\mathcal{N}'_j$ denote the non-faulty nodes in the neighborhood of node $j$, i.e., $\mathcal{N}'_j = J' \cap \mathcal{N}_j$. The set of Byzantine neighbors can be defined as $\mathcal{N}_b = \mathcal{N}_j \setminus \mathcal{N}'_j$. If node $j$ is a non-faulty node, we can write the update as

$$[w_j(t+1)]_k = \frac{1}{|\mathcal{N}_j| - 2b + 1} \sum_{i \in \mathcal{N}'_j \setminus J} [w_i(t)]_k$$

$$-\rho(t)[\nabla \tilde{f}(\bar{w}(t+1, k))^T, [w(t)]_k, S_j)_k].$$

(19)

We make an observation that the update of $[w_j]_k$ does not depend on $[w_{ij}]_k$ for $p \neq k$. So we only show that all nodes have consensus in the dimension $k$ and we can conclude that all nodes have consensus on all dimensions. Note that $\mathcal{N}_b(t)$ could contain both Byzantine and non-faulty nodes. Form a vector $\Omega(t) \in \mathbb{R}^{|J'| \times 1}$ satisfying $[\Omega(t)]_j = [w_j(t)]_k \forall j \in J'$. The first thing we want to show here is that the update of all non-faulty nodes can be written in a matrix form which contains only non-faulty nodes, i.e.,

$$\Omega(t+1) = Y(t)\Omega(t) - \rho(t)G(t).$$

(20)

where $G(t) \in \mathbb{R}^{|J'| \times 1}$ and $[G(t)]_j = \rho(t)[\nabla \tilde{f}(\bar{w}(t + 1, k), [w(t)]_k, S_j)_k].$

One of two cases can happen during each iteration, (a) $\mathcal{N}_b(t) \cap \mathcal{N}_b \neq \emptyset$ or (b) $\mathcal{N}_b(t) \cap \mathcal{N}_b = \emptyset$. For case (a), since $|\mathcal{N}_b| \leq \theta$ and $|\mathcal{N}_b| = b$, we know that $\mathcal{N}_b(t) \cap \mathcal{N}_b \neq \emptyset$. Similarly, $\mathcal{N}_b(t) \cap \mathcal{N}_b \neq \emptyset$. Then $\exists i \in \mathcal{N}_b(t) \cap \mathcal{N}_b(t)$ and $l_j(t) \in \mathcal{N}_b(t) \cap \mathcal{N}_b(t)$ satisfying $[w_{ij}(t)]_k < [w_i(t)]_k < [w_j(t)]_k$ for any $i \in \mathcal{N}_b(t)$. So that for each $i \in \mathcal{N}_b(t) \cap \mathcal{N}_b(t)$, $\exists i \in \mathcal{N}_b(t) \cap \mathcal{N}_b(t)$ satisfying $[w_i(t)]_k = \theta_i(t)[w_j(t)]_k + (1 - \theta_i(t))[w_j(t)]_k$. We separate the update with $\mathcal{N}_b$ and $\mathcal{N}_b$,.

$$[w_j(t+1)]_k = \frac{1}{|\mathcal{N}_j| - 2b + 1} \left([w_j(t)]_k + \sum_{i \in \mathcal{N}_b(t) \setminus \mathcal{N}_b(t)} [w_i(t)]_k\right)$$

$$+ \sum_{i \in \mathcal{N}_b(t) \setminus \mathcal{N}_b(t)} [w_i(t)]_k + \rho(t)[\nabla \tilde{f}(w_j(t), S_j)]_k$$

$$= \frac{1}{|\mathcal{N}_j| - 2b + 1} \left([w_j(t)]_k + \sum_{i \in \mathcal{N}_b(t) \setminus \mathcal{N}_b(t)} [w_i(t)]_k\right)$$

$$+ \sum_{i \in \mathcal{N}_b(t) \setminus \mathcal{N}_b(t)} \left(\theta_i(t)[w_j(t)]_k + (1 - \theta_i(t))[w_j(t)]_k\right)$$

$$+ \rho(t)[\nabla \tilde{f}(w_j(t), S_j)]_k.$$.

Observe that in (21) we do not use any $w$ from Byzantine nodes to update $w_j$. In this way, we can write the update in the matrix form containing only non-faulty nodes as in (20). The elements of matrix $Y(t)$ can be written as

$$[Y(t)]_{ji} = \begin{cases} 
\frac{|\mathcal{N}_j| - 2b + 1}{|\mathcal{N}_j| - 2b + 1} & i \in \mathcal{N}_j \cap \mathcal{N}_j(t) \\
\frac{\theta_i(t)}{|\mathcal{N}_j| - 2b + 1} & i = j \\
\sum_{i \in \mathcal{N}_b(t) \setminus \mathcal{N}_j(t)} \frac{1 - \theta_i(t)}{|\mathcal{N}_j| - 2b + 1} & i = l_j(t) \\
0 & \text{else.}
\end{cases}$$

(22)

Case (b) can be viewed as a special case of case (a) where we keep only the first, second and last rows of (22) because $\mathcal{N}_b(t) \cap \mathcal{N}_j(t) = \emptyset$. Note that since the choices of $s_j(t)$ and $l_j(t)$ are generally not unique, the matrix $Y(t)$ is also not unique.

More details on the formulation and properties of matrix $Y(t)$ can be found in (31). Here we mainly focus on the property that $Y(t)$ is row stochastic and the fact that the product of row stochastic matrices is also row stochastic. According to (20), if we start from some time step $t_0$.

$$\Omega(t_0 + 1) = Y(t_0)\Omega(t_0) - \rho(t_0)G(t_0)$$

$$\Omega(t_0 + 2) = Y(t_0 + 1)\Omega(t_0 + 1) - \rho(t_0 + 1)G(t_0 + 1)$$

$$= Y(t_0 + 1)Y(t_0)\Omega(t_0) - Y(t_0 + 1)\rho(t_0)G(t_0) - \rho(t_0 + 1)G(t_0 + 1)$$

$$\ldots$$

$$\Omega(t + 1) = Y(t)\Omega(t) - \rho(t)G(t)$$

$$= Y(t)Y(t - 1) \cdots Y(t_0)\Omega(t_0)$$

$$- \sum_{\tau = t_0}^{t - 1} Y(t)Y(t - 1) \cdots Y(t_0 + 1)\rho(\tau)G(\tau)$$

$$- \rho(t)G(t).$$

(23)

Define a transition matrix $\Phi(t, t_0) = Y(t)Y(t - 1) \cdots Y(t_0)$. Then the relationship between starting point $\Omega(t_0)$ and any time point $\Omega(t)$ can be written as

$$\Omega(t + 1) = \Phi(t, t_0)\Omega(t_0) - \sum_{\tau = t_0}^{t} \Phi(t, \tau + 1)\rho(\tau)G(\tau).$$

(24)

In order to show the convergence of the algorithm, we introduce some properties of transition matrix $\Phi(t, t_0)$. Let $\pi$ denote a stochastic vector.

Lemma 4: [30] Let Assumption 2 hold, then for any starting point $t_0$, $\exists \pi(t_0)$ that

$$\lim_{t \to \infty} \Phi(t, t_0) = 1\pi^T(t_0).$$

(25)

Note that transition matrix is the product of matrices $Y(t)$. Lemma 4 shows that the product of row stochastic matrices $Y(t)$ converges to a steady state where all rows of $\Phi$ are stochastic and identical. Let $\psi$ be the total number of reduced graphs we can generate from $G$. Let $\nu = \psi |J'|$. We denote $\max_{j \in J} |N_j| \leq N_{\max}$. Then let $\mu = 1 - \frac{1}{(2N_{\max} - 2b + 1)^\nu}$.

Lemma 5: [30] Let Assumption 2 hold, $\forall t \geq t_0$,

$$[\Phi(t, t_0)]_{ji} - [\pi(t_0)]_i \leq \mu^{t_0 + i}.$$.

(26)

Lemma 5 gives the convergence rate of $\Phi$. Next we will show that consensus can be guaranteed as $t \to \infty$. If we initiate the algorithm from $\Omega(0)$, according to (24).

$$\Omega(t + 1) = \Phi(t, 0)\Omega(0) - \sum_{\tau = 0}^{t} \Phi(t, \tau + 1)\rho(\tau)G(\tau).$$

(27)

Assume that all nodes stop computing local gradients at time step $t$ and use $G(t + T) = 0$ for $T \geq 0$. We define a vector
Algorithm 2 ByRDiE with switching coordinate

Input: \( b, S_1, S_2, \ldots, S_M \)
1: Initiate \( w_j(0) = 0 \), \( j = 1, 2, \ldots, |J'| \)
2: for \( j = 1, 2, 3, \ldots, |J'| \) do In parallel
3: for \( t = 1, 2, 3, \ldots \) do
4: for \( k = 1, 2, 3, \ldots, P \) do
5: Broadcast \( [w_j(t)]_k \) to all neighbors in \( N_j \)
6: Receive \( [w_i(t)]_k \) for \( i \in N_j \)
7: Sort \( \{[w_i(t)]_k\} \) in increasing order \( \forall i \in N_j \)
8: Define \( \mathcal{N}_j^*(t) = \arg \min_{X : X \in \mathcal{N}_j, |X| = b} \sum [w_j(t)]_k \)
9: Define \( \mathcal{N}_j^+(t) = \arg \max_{X : X \in \mathcal{N}_j, |X| = b} \sum [w_j(t)]_k \)
10: Define \( \mathcal{N}_j^0(t) = \mathcal{N}_j \setminus \mathcal{N}_j^0(t) \setminus \mathcal{N}_j^0(t) \)
11: Update \( w_j \) as \( [w_j(t + 1)]_k = \frac{1}{|\mathcal{N}_j| - 2b + 1} \sum_{i \in \mathcal{N}_j^0(t), j} [w_i(t)]_k \)
12: \(-\rho(t) [\nabla f(\bar{w}(t + 1, k, [w(t)]_k), S_j)_k] \)
13: end for
14: end for
15: end for
Output: \( w_j, \forall j \in J' \)

under this assumption:

\[
V_k(t) = \lim_{T \to \infty} \Omega(t + T + 1)
= \lim_{T \to \infty} \Phi(t + T, 0)\Omega(0) - \lim_{T \to \infty} \sum_{\tau = 0}^{t+T} \Phi(t + T, \tau)\rho(\tau)G(\tau)
= 1\pi^T(0)\Omega(0) - \sum_{\tau = 0}^{t+T} 1\pi^T(\tau)\rho(\tau)G(\tau)
= 1\pi^T(0)\Omega(0) - \sum_{\tau = 0}^{t+T} 1\pi^T(\tau)\rho(\tau)G(\tau).
\]

Observe that all elements in \( V_k(t) \) are identical. Let \( v_k(t) = [V_k(t)]_j \) for some \( j \in J' \). If we can show that \( [w_j(t)]_k \to v_k(t) \) for any \( k \), then we can conclude that all non-faulty nodes achieve consensus.

Lemma 6: Let Assumption 12 and 3 hold. Form a sequence \( V(t) \) satisfying \( [V(t)]_k = v_k(t) \) for all \( j \in J' \). Then \( w_j(t) \to V(t) \) as \( t \to \infty \).

Lemma 7 shows that the consensus can be achieved. The proof can be found in Appendix C. Next we show that \( v_k(t) \to [w^*]_k \). We define a sequence \( Q(q) \) where \( Q(0)_k = v_k(0) \) for \( 1 \leq k \leq P \). Then for \( q = Pt + k \neq 0 \), the sequence \( Q(q) \) is obtained by replacing \( Q(q - 1)_k \) with \( v_k(t) \) while other dimensions remaining the same. From (28), we know that

\[
v_k(t + 1) = v_k(t) - \rho(t) \sum_{i=1}^{\frac{|J'|}{P}} [\pi(t + 1)]_i [\nabla f(w_i(t), S_i)]_k. \tag{29}
\]

Similarly we define a sequence \( \eta(q) = \rho(t) \) for \( Pt \leq q < (P + 1)t \). Note that this sequence also has the property \( 0 < \eta(q + 1) \leq \eta(q) \), \( \sum_{q=1}^{\infty} \eta(q) = \infty \) and \( \sum_{q=1}^{\infty} \eta^2(q) < \infty \). So the update of sequence \( Q(q) \) is

\[
Q(q + 1) = Q(q) - \eta(q) \sum_{i=1}^{\frac{|J'|}{P}} [\pi(t + 1)]_i [\nabla f(w_i(t), S_i)]_k e_k \tag{30}
\]

where \( e_k \) is a vector that is zero at every dimension except for \( |e_k)_k = 1 \). Since \( t \geq 1 \) and \( 1 \leq k \leq P, q = Pt + k \) can be uniquely determined. The purpose of forming sequence \( Q(q) \) is to reflect the change in \( v_j(t) \) right after each update in one dimension while the sequence \( w_j(t) \) can only show the change after every \( P \) updates.

Lemma 7: Let Assumption 12 and 3 hold. When \( N \to \infty \), \( Q(q) \to w^* \) as \( q \to \infty \).

Recall that \( [Q(q)]_k = v_k(t) \) for \( q = Pt + k \) and \( v_k(t) = [V(t)]_k \), which leads to \( V(t) \to w^* \). The proof of Lemma 7 can be found in Appendix D. Together with Lemma 3, we have shown that \( w_j \to V(t) \) and \( V(t) \to w^* \). So the proof of Theorem 2 is complete.

C. Comparison of the two implementations

The ByRDiE algorithm combines two tasks together in both implementations: consensus and optimization. We have shown in the theorems, when the iteration indicators \( r \) and \( t \) go to infinity, both tasks are fulfilled by both implementations. The reason to say that we are introducing two ways of implementation instead of calling them different algorithms is that they can be considered as a special case to each other. Recall implementing ByRDiE with exact line search we need to set up a threshold for consensus at each dimension. So if we set the threshold large enough that it can be satisfied by one message passing, then the process is identical to implementing ByRDiE with switching coordinate. Moreover, for coordinate descent, usually there is no fixed order to choose which dimension to update first and the dimension can also be chosen randomly. As a result, exact line search can be viewed as implementing ByRDiE with switching coordinate but sequentially picking the same dimension several times. However, before all nodes agree on the global minimum, implementing ByRDiE with exact line search prioritizes consensus while with switching coordinate prioritizes optimization. We can observe from the process of exact line search that the whole network achieves consensus before switching to another dimension. So if we need \( \Gamma \) rounds of updates to achieve consensus for when implementing ByRDiE with exact line search, we need roughly \( \Gamma P \) rounds of updates to achieve consensus with switching coordinate because each dimension is updated only once in every \( P \) updates. So when \( P \) is large, the difference will be obvious. On the other hand, when implementing ByRDiE with switching coordinate, it only takes \( P \) rounds to update all dimensions of local variables while exact line search can only update roughly \( P \) coordinates with the same amount of communication. This trade-off should be considered according to the nature of different applications.

IV. Numerical results

In this section, we apply our algorithm on the MNIST8M handwriting digits dataset to show two facts: (i) the classic
distributed learning algorithms fail when there is Byzantine failure in the network; (ii) our algorithm is indeed Byzantine-resilient. MNIST8M dataset contains 8,100,000 images of handwriting digits from '0' to '9' of which 8,000,000 images are for training and 100,000 images are for testing. Each digit is stored in 28 × 28 pixels. Since the main purpose of this numerical analysis is to show the performance difference under Byzantine settings rather than pursuing the highest classification accuracy, we model the analysis as a simple binary classification problem, i.e., distinguishing digit '5' and '8', which are the two most inseparable sets. We distribute training samples into 200 nodes with 3500 digit '5' and 3500 digit '8' each node. Test set is of size 40,000 with equal number of both digits. Pick 10 nodes to be Byzantine nodes which will broadcast random values to their neighbors in each iteration. Only none-faulty nodes will be tested after training and we only consider the 190 none-faulty nodes for classification accuracy.

We use the well-understood SVM to find the classifier. We perform training with our algorithm and compare our result with another well-understood distributed optimization method: distributed gradient descent (DGD). Since the dimensionality of each image is 784 and the network has 200 nodes, the network does not satisfy the strong constrains like in [26]. There is no existing distributed method that solves this classification problem in the presence of Byzantine failure to the best of our knowledge. So instead of comparing to existing distributed methods, we use centralized SVM as the baseline of comparison.

We run three rounds of tests: (1) Collecting all training samples together and run centralized SVM; (2) SVM via DGD with Byzantine failures in the network; (3) SVM via our Byzantine-Resilient Distributed coordinate Descent (ByRDiE) with Byzantine failures in the network. Two measurements should be taken into account: accuracy and consensus. Accuracy is measured by the rate of classifying test samples correctly. We take the $\ell_2$-norm to represent the difference between any two local classifiers. Consensus is measured by taking the maximum among these norm of differences. By the nature SGD and measurement of comparison, implementing ByRDiE with switching coordinate is more suitable for this test.

The result is shown in Table I, Fig.1 and Fig.2. As a baseline, centralized SVM achieves the highest accuracy of 93.96%. Classic DGD has an accuracy as low as 50% for giving all test samples the same label. Meanwhile, consensus measure of DGD increases exponentially fast due to the false information from Byzantine nodes, which indicates that the non-resilient distributed method fails when there are Byzantine failures in the network. On the other hand, a node can choose not to cooperate with others and perform training only on local data, which achieves an accuracy of 90.28%. The sharp decrease of consensus measure and the accuracy of 91.46% show that ByRDiE can indeed improve the classification performance by cooperation among all nodes while staying resilient to Byzantine failure.

| Algorithm       | Sample size | Accuracy | Consensus |
|-----------------|-------------|----------|-----------|
| Centralized SVM | 700000 × 1  | 92.30%   | N/A       |
| SVM via DGD     | 3500 × 200  | 50%      | NO        |
| SVM via ByRDiE  | 3500 × 200  | 91.46%   | YES       |
| Local SVM       | 3500 × 1    | 90.28%   | N/A       |

![Fig. 1. The maximum norm of difference between two local classifiers is used as the measure of consensus. It is shown in the upper graph that the norm of difference for classic DGD increases exponentially when there are Byzantine failures in the network, which implies that the method fails because local classifiers do not converge. The lower graph shows that the maximum difference between local classifiers decreases over time, which shows that ByRDiE can indeed achieve consensus when there are Byzantine failures in the network.](image1)

![Fig. 2. The accuracy of classification on the testset is used as the measure of performance of algorithms. Red line is the accuracy of centralized method with all training samples, which is the baseline of performance. Blue line is the accuracy using ByRDiE when there are Byzantine failures in the network. Blue line is the accuracy of using only local data for training. The graph shows that cooperation among all nodes has a better performance even there are Byzantine failures in the network.](image2)

V. CONCLUSION

This paper proposes an algorithm that solves high dimensional distributed learning problem. Theoretical guarantees are given along with the proof. There are two main contributions in our work. First is that our algorithm converges the true minimum of the learning problem with high probability under the assumption that the training samples are i.i.d.. This is oppose to the existing works in which algorithms are designed to converge to a weighted combination of local losses, which is usually not an objective of machine learning tasks. The second contribution is that existing works make strong assumptions.
on the network topology depending on the depending on the dimension of training data. When the dimension of the problem increases, the constrain becomes impractical. Our assumption of network topology does not depend on the dimension of training data, which makes the algorithm scale well for high dimensional problems.

**APPENDIX A**

**Proof of Lemma 2**

First we define a function \( \hat{F} \) as

\[
\hat{F}(w, S) = \sum_{j=1}^{M} \alpha_j \hat{f}(w, S_j)
\]

\[
= R(w) + \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} \alpha_j \ell(w, (x_{jn}, y_{jn}))
\]

for some \( \alpha_j \geq 0 \) and \( \sum_{j=1}^{M} \alpha_j = 1 \). The difference between \( \hat{F} \) and its expectation is

\[
(\hat{F}(w, S) - \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \hat{F}(w, (x_{jn}, y_{jn})))
\]

\[
= \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} \alpha_j \ell(w, (x_{jn}, y_{jn}))
\]

\[
- \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} \alpha_j \ell(w, (x_{jn}, y_{jn}))
\]

\[
= \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} (\alpha_j \ell(w, (x_{jn}, y_{jn})))
\]

\[
- \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \ell(w, (x_{jn}, y_{jn})).
\]

Assumption 3 ensures that \( 0 \leq \alpha_j \ell(w, (x_{jn}, y_{jn})) \leq \alpha_j C \). If each sample \((x_{jn}, y_{jn})\) picked from \( D \) is i.i.d, then according to the Hoeffding’s inequality [33],

\[
\mathbb{P}(\frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} \alpha_j \ell(w, (x_{jn}, y_{jn}))
\]

\[
- \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \ell(w, (x_{jn}, y_{jn})) | \geq \epsilon)
\]

\[
\leq 2 \exp\left(-\frac{2N \epsilon^2}{\sum_{j=1}^{M} \alpha_j C^2}\right).
\]

Because

\[
\mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \hat{F}(w, (x_{jn}, y_{jn}))
\]

\[
= R(w) + \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \sum_{j=1}^{M} \alpha_j \ell(w, (x_{jn}, y_{jn}))
\]

\[
= R(w) + \sum_{j=1}^{M} \alpha_j \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \ell(w, (x_{jn}, y_{jn}))
\]

\[
= R(w) + \sum_{j=1}^{M} \alpha_j \mathbb{E}_{(x,y) \sim D} \ell(w, (x, y))
\]

\[
= \mathbb{E}_{(x,y) \sim D} \ell(w, (x, y)),
\]

we can replace \( \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \hat{F}(w, (x_{jn}, y_{jn})) \) in [32] by \( \mathbb{E}_{(x,y) \sim D} \ell(w, (x, y)) \). Then we have

\[
\mathbb{P}(\mathbb{H}(\hat{w}, S) - \mathbb{E}_{(x,y) \sim D} \ell(\hat{w}, (x, y))) \geq \epsilon)
\]

\[
\leq 2 \exp\left(-\frac{2N \epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}\right).
\]

As \( N \to \infty \), [35] shows that \( \hat{F}(w, S) \to \mathbb{E}_{(x,y) \sim D} \ell(w, (x, y)) \) a.s. for any fixed \( w \). By the definition of \( h \) and \( H \), inequality [35] can be written as

\[
\mathbb{P}(\mathbb{H}(\hat{w}, S) - \mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y))) \geq \epsilon)
\]

\[
\leq 2 \exp\left(-\frac{2N \epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}\right).
\]

We take a union bound and conclude that with probability at least \( 1 - 4 \exp\left(-\frac{2N \epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}\right) \), we have

\[
|H(\hat{w}, S) - \mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y))| \leq \epsilon
\]

and

\[
H(w^*, S) - \mathbb{E}_{(x,y) \sim D} h(w^*, (x, y)) \leq \epsilon.
\]

Let \( \delta = 4 \exp\left(-\frac{2N \epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}\right) \). Then with probability at least \( 1 - \delta \) we have

\[
|H(\hat{w}, S) - \mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y))| \leq \sqrt{\frac{C^2 \ln \frac{4}{\delta} \sum_{j=1}^{M} \alpha_j^2}{2N}}
\]

and

\[
H(w^*, S) - \mathbb{E}_{(x,y) \sim D} h(w^*, (x, y)) \leq \epsilon.
\]
Next we prove (10) by contradiction. Assume that when (35) is simultaneously true for all \( w \), (10) is not true and there exists some \( N_0 \) satisfying

\[
|\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y)) - \mathbb{E}_{(x,y) \sim D} h(w^*, (x, y))| \leq \epsilon'.
\]  

(42)

Since \( w^{**} \) is the minimum of \( h \), we know that

\[
\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y)) - \mathbb{E}_{(x,y) \sim D} h(w^{**}, (x, y)) = \epsilon'.
\]  

(43)

Then from (40) and (41) we know that

\[
\mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y)) - \sqrt{\frac{C^2 \ln \frac{4}{\delta} \sum_{j=1}^{M} \alpha_j^2}{2N_0}} \leq H(\hat{w}, S)
\]  

(44)

\[
H(w^{**}, S) \leq \mathbb{E}_{(x,y) \sim D} h(w^{**}, (x, y)) + \sqrt{\frac{C^2 \ln \frac{4}{\delta} \sum_{j=1}^{M} \alpha_j^2}{2N_0}}.
\]  

(45)

Substituting (43) into (45) we have

\[
H(w^{**}, S) \leq \mathbb{E}_{(x,y) \sim D} h(\hat{w}, (x, y)) - \epsilon' + \sqrt{\frac{C^2 \ln \frac{4}{\delta} \sum_{j=1}^{M} \alpha_j^2}{2N_0}}.
\]  

(46)

Step (a) is true because of (42). Plugging (44) into (46) we have

\[
H(w^*, S) < H(\hat{w}, S).
\]  

(47)

By the definition of \( \hat{w} \), we know that \( H(\hat{w}, S) \leq H(w^{**}, S) \), which contradicts with (47). So (10) must hold true. Here we only allow \( 0 < \delta < 1 \) because \( 1 - \delta \) represents a probability. So we conclude that (10) is true with probability at least \( 1 - \delta \), which is the Lemma 2.

**APPENDIX B**

**PROOF OF LEMMA 3**

Similarly in (32) and (34) from the proof of lemma 2 we observe that

\[
\mathbb{E}_{(x,y) \sim D} F_k(\bar{w}(r, k, w'), S) = \mathbb{E}_{(x,y) \sim D} f(\bar{w}(r, k, w'), (x, y))
\]  

(48)

and

\[
F_k(\bar{w}(r, k, w'), S) = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} [\beta_j(r)k\ell(w, (x_{jn}, y_{jn})) - \mathbb{E}_{(x_{jn}, y_{jn}) \sim D} \ell(w, (x_{jn}, y_{jn}))]
\]  

(49)

The main difference from lemma 3 to lemma 2 is that we need to show the convergence simultaneously true in all dimensions during all iterations. Next we are going to show that as \( N \to \infty \), \( F_k(\bar{w}(r, k, w'), S) \to \mathbb{E}_{(x,y) \sim D} f(\bar{w}(r, k, w'), (x, y)) \) a.s. simultaneously true for all \( \bar{w} \).

We denote the smallest sphere in \( \mathbb{R}^P \) that contains \( W \) as \( W' \). We denote the radius of \( W' \) as \( \Gamma \). According to (34), we can cover \( W \) with at most \( P \sqrt{P} \ln P(\frac{\epsilon}{\gamma})^{P-1} \) equal sized spheres in \( \mathbb{R}^P \) with radius \( \gamma \). Let \( d = P \sqrt{P} \ln P(\frac{\epsilon}{\gamma})^{P-1} \). If we denote the center of one sphere as \( w_1 \), we know that \( w_1 \) satisfies

\[
\mathbb{P}(|F_k(w_1, S) - \mathbb{E}_{(x,y) \sim D} f(w_1, (x, y))| \geq \epsilon) \leq 2 \exp(- \frac{2N\epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}).
\]  

(50)

When we pick a point \( w_2 \) within the sphere, i.e., \( \|w_1 - w_2\| \leq \gamma \), we know that

\[
|F_k(w_1, S) - F_k(w_2, S)| \leq L'\gamma
\]  

(51)

and

\[
|\mathbb{E}_{(x,y) \sim D} f(w_1, (x, y)) - \mathbb{E}_{(x,y) \sim D} f(w_2, (x, y))| \leq L'\gamma
\]  

(52)

So \( w_2 \) satisfies

\[
\mathbb{P}(|F_k(w_2, S) - \mathbb{E}_{(x,y) \sim D} f(w_2, (x, y))| \geq \epsilon + 2L'\gamma) \leq 2 \exp(- \frac{2N\epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}).
\]  

(53)

Since there are \( d \) spheres and \( w_1 \) is arbitrarily chosen from them, there exists a union bound

\[
\mathbb{P}(\bigcup_{w \in W} (|F_k(w, S) - \mathbb{E}_{(x,y) \sim D} f(w, (x, y))| \geq \epsilon + 2L'\gamma)) \leq 2d \exp(- \frac{2N\epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}).
\]  

(54)

This is equivalent to

\[
\mathbb{P}(\bigcap_{w \in W} (|F_k(w, S) - \mathbb{E}_{(x,y) \sim D} f(w, (x, y))| \leq \epsilon + 2L'\gamma)) \geq 1 - 2d \exp(- \frac{2N\epsilon^2}{C^2 \sum_{j=1}^{M} \alpha_j^2}),
\]  

(55)
Let \( \{\alpha_j\} = \arg\max_{\{\beta_j(r)\}_{k+r-1} \geq 1} \sum_{j=1}^{M} [\beta_j(r)]_k^2 \), then we have

\[
P(\{w \in W \mid \sum_{(x,y) \sim D} |F_k(w, S) - E_{(x,y) \sim D} f(w, (x, y))| \leq \epsilon + 2L\gamma \}) \\
\geq 1 - \frac{2N\epsilon^2}{C^2 \sum_{j=1}^{M} [\beta_j(r)]_k^2}.
\]

(56)

Note that \( \gamma \) and \( \epsilon \) can both be arbitrarily small. So when \( N \rightarrow \infty, F_k(w, S) \rightarrow E_{(x,y) \sim D} f(w, (x, y)) \) a.s. \( \forall w \in W \) simultaneously. With a similar argument in Appendix A, one can show that

\[
\min_{w' \in W} F_k(\bar{w}(r, k, w'), S) \rightarrow \min_{w' \in W} E_{(x,y) \sim D} f(\bar{w}(r, k, w'), (x, y)).
\]

(57)

Combine the definition of \( w(r) \) with (57). Lemma 3 is proved.

**APPENDIX C**

**PROOF OF LEMMA 6**

From (28),

\[
V_k(t + 1) = 1\pi^T(0)\Omega(0) \\
- \sum_{\tau=0}^{t-1} 1\pi^T(\tau + 1)\rho(\tau)G(\tau) - 1\pi^T(t + 1)\rho(t)G(t).
\]

(58)

Then \( v_k(t + 1) \) can be written as

\[
v_k(t + 1) = \sum_{i=1}^{\lfloor J \rfloor} [\pi(0)]_i [w_i(0)]_k \\
- \sum_{\tau=0}^{t-1} \rho(\tau) \sum_{i=1}^{\lfloor J \rfloor} [\pi(\tau + 1)]_i [\nabla f(w_i(\tau), S)]_k \\
- \rho(t) \sum_{i=1}^{\lfloor J \rfloor} [\pi(t + 1)]_i [\nabla f(w_i(t), S)]_k.
\]

(59)

Recall that \( [\Omega(t)]_j = [w_j(t)]_k \), then from (27)

\[
[w_j(t + 1)]_k = \sum_{i=1}^{\lfloor J \rfloor} [\pi(0)]_i [w_i(0)]_k \\
- \sum_{\tau=0}^{t-1} \rho(\tau) \sum_{i=1}^{\lfloor J \rfloor} [\pi(\tau + 1)]_i [\nabla f(w_i(\tau), S)]_k \\
- \rho(t) [\nabla f(w_j(t), S)]_k
\]

(60)

where we used the fact that \( \Phi(t, t + 1) = I \).

\[
[w_j(t + 1)]_k - v_k(t + 1) = \\
\sum_{i=1}^{\lfloor J \rfloor} ([\pi(0)]_i - [\pi(0)]_i) [w_i(0)]_k \\
- \sum_{\tau=0}^{t-1} \rho(\tau) \sum_{i=1}^{\lfloor J \rfloor} ([\pi(\tau + 1)]_i - [\pi(\tau + 1)]_i) [\nabla f(w_i(\tau), S)]_k \\
- \rho(t) \sum_{i=1}^{\lfloor J \rfloor} [\pi(t + 1)]_i [\nabla f(w_j(t), S)]_k - [\nabla f(w_i(t), S)]_k
\]

(61)

where the third term on the r.h.s. comes from the fact that

\[
\sum_{i=1}^{\lfloor J \rfloor} [\pi(t + 1)]_i = 1.
\]

If Assumption 1 and 3 hold, we can always find two scalars \( C_w \) and \( L \) satisfying \( \forall j \in J', |w_j(0)| \leq C_w \) and \( |\nabla f(w_j, S)|_k \leq L \). Then we have

\[
[w_j(t + 1)]_k - v_k(t + 1) \\
\leq \sum_{i=1}^{\lfloor J \rfloor} ([\pi(0)]_i - [\pi(0)]_i) [w_i(0)]_k \\
+ \sum_{\tau=0}^{t-1} \rho(\tau) \sum_{i=1}^{\lfloor J \rfloor} [\pi(\tau + 1)]_i [\nabla f(w_i(\tau), S)]_k \\
+ \rho(t) \sum_{i=1}^{\lfloor J \rfloor} [\pi(t + 1)]_i [\nabla f(w_j(t), S)]_k - [\nabla f(w_i(t), S)]_k \\
\leq |J'|C_w\mu^{\frac{t+1}{1}} + |J'|L' \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} + 2\rho(t)L'.
\]

(62)

The convergence of the term \( |J'|L' \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} \) is not obvious. It can be further expanded as

\[
|J'|L' \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} = |J'|L' \left( \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} + \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} \right)
\]

\[
|J'|L' \left( \rho(0) \mu^{\frac{t+1}{1}} + \rho(t) \mu^{\frac{t+1}{1}} \right) \\
\leq |J'|L' \left( \rho(0) \mu^{\frac{t+1}{1}} \left( 1 - \frac{\mu}{\frac{t+1}{1}} \right) + \rho(t) \mu^{\frac{t+1}{1}} \left( 1 - \frac{\mu}{\frac{t+1}{1}} \right) \right).
\]

(63)

Taking limit on both side,

\[
\lim_{t \to \infty} |J'|L' \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} \leq |J'|L' \lim_{t \to \infty} \left( \rho(0) \mu^{\frac{t+1}{1}} \left( 1 - \frac{\mu}{\frac{t+1}{1}} \right) + \rho(t) \mu^{\frac{t+1}{1}} \left( 1 - \frac{\mu}{\frac{t+1}{1}} \right) \right)
\]

\[
= 0.
\]

(64)

Since \( |J'|L' \sum_{\tau=0}^{t-1} \rho(\tau)\mu^{\frac{t+1}{1}} \geq 0 \), we conclude that

\[
\lim_{t \to \infty} |w_j(t + 1)]_k - v_k(t + 1) = 0.
\]

(65)

Here we have shown that \( [w_j(t)]_k \) converges to the same value in dimension \( k \). Since the choice of \( k \) is trivial, we can form
a sequence \( V(t) \) satisfying \([V(t)]_k = v_k(t)\). Then we have \( w_j(t) \to V(t) \) for \( j \in J' \) as \( t \to \infty \).

**APPENDIX D
PROOF OF LEMMA 7**

To prove the lemma, first let us modify the update \( \text{(30)} \) a little bit,

\[
Q(q + 1) = Q(q) - \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), S_i)]_k e_k
+ \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), S_i)]_k e_k
- \eta(q) \sum_{i=1}^{[J']} [\nabla f(w_i(t), S_i)]_k e_k.
\]

Then we rearrange the terms

\[
(\eta(q) - \eta^2(q)L) [\nabla f(Q(q), (x,y))]_k^2
\leq f(Q(q), (x,y)) - f(Q(q + 1), (x,y))
+ \frac{L}{2} [E(q)]_k^2 + (1 - L\eta(q)[\nabla f(Q(q), (x,y))]_k) [E(q)]_k.
\]

Take summation on both side with \( q \to \infty \), we have

\[
\sum_{q=0}^{\infty} (\eta(q) - \frac{\eta^2(q)L}{2}) [\nabla f(Q(q), (x,y))]_k^2
\leq \sum_{q=0}^{\infty} (f(Q(q), (x,y)) - f(Q(q + 1), (x,y)))
+ \sum_{q=0}^{\infty} \frac{L}{2} [E(q)]_k^2 + \sum_{q=0}^{\infty} (1 - L\eta(q)[\nabla f(Q(q), (x,y))]_k) [E(q)]_k.
\]

(71)

On the l.h.s. of (71), we know that \( \sum_{q=0}^{\infty} \eta(q) = \infty \). So if the r.h.s. does not go to infinity, we can conclude that \([\nabla f(Q(q), (x,y))]_k^2 \to 0\). First term on the r.h.s. can be bounded as

\[
\sum_{q=0}^{\infty} (f(Q(q), (x,y)) - f(Q(q + 1), (x,y))) = f(Q(0), (x,y)) - f(Q(\infty), (x,y)) \leq f(Q(0), (x,y)) - f(w^*, (x,y))
\]

Both of the rest two terms include \([E(q)]_k = \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), (x,y))]_k\), when Assumption 7 is true, by the definition of \( L' \), we know that

\[
[\nabla f(Q(q), (x,y))]_k - [\nabla f(w_i(t), S_i)]_k \leq L' [Q(q) - w_i(t)]
\]

Therefore the definition of \( E(q) \) as

\[
E(q)_k = \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), (x,y))]_k - [\nabla f(w_i(t), S_i)]_k
\]

By the definition of \( Q(q) \) and (62), we can further bound \( E(q)_k \) as

\[
E(q)_k = \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), (x,y))]_k - [\nabla f(w_i(t), S_i)]_k
\]

\[
\leq \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), S_i)]_k - [\nabla f(w_i(t), S_i)]_k
\]

\[
\leq \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), (x,y))]_k - [w_i(t)]_k
\]

\[
\leq \eta(q) \sum_{i=1}^{[J']} [\nabla f(Q(q), (x,y))]_k - [w_i(t)]_k
\]

(74)

where we define \( i'(k, t) = \arg \max_{i \in J'} ||Q(q) - w_i(t)||_k \). Considering that \( \eta(q) = \rho(t) \) for \( q = Pt + k \), \( \sum_{q=0}^{\infty} \eta(q) \rho(t) < \infty \).

So \( \sum_{q=0}^{\infty} [E(q)]_k < \infty \). Then together with (72), we can interpret (71) as

\[
\sum_{q=0}^{\infty} (\eta(q) - \frac{\eta^2(q)L}{2}) [\nabla f(Q(q), (x,y))]_k^2 < \infty.
\]

(75)

This leads to \([\nabla f(Q(q), (x,y))]_k \to 0\). The result is true for any dimension \( k \) so that \( \nabla f(Q(q), (x,y)) \to 0 \). Since \( f(Q(q), (x,y)) \) is strongly convex, \( \nabla f(Q(q), (x,y)) \to 0 \) also means that \( Q(q) \to w^* \), which proves Lemma 7.
REFERENCES

[1] F. Sebastiani, “Machine learning in automated text categorization,” ACM computing surveys (CSUR), vol. 34, no. 1, pp. 1–47, 2002.
[2] S. Kotzisantis, “Supervised machine learning: A review of classification techniques,” Informatica J., vol. 31, no. 3, pp. 249–268, 2007.
[3] Y. Bengio, “Learning deep architectures for AI,” Foundations and trends in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.
[4] M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of machine learning. Cambridge, MA, USA: MIT press, 2012.
[5] J. B. Predd, S. R. Kulkarni, and H. V. Poor, “Distributed learning in wireless sensor networks,” IEEE Sig. Process. Mag., vol. 22, no. 4, pp. 56–69, 2006.
[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Foundations and Trends in Machine Learning, vol. 3, no. 1, pp. 1–122, 2011.
[7] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, “Byzantine fault tolerance, from theory to reality,” in Proc. Int. Conf. Computer Safety, Reliability and Security, 2003, pp. 235–248.
[8] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona, “The real byzantine generals,” in Proc. 23rd Digital Avionics Systems Conf., vol. 2, IEEE, 2004, pp. 6-D.
[9] L. Lampert, R. Shahak, and M. Pease, “The byzantine generals problem,” ACM Trans. Programming Languages and Systems, vol. 4, no. 3, pp. 382–401, 1982.
[10] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” J. ACM, vol. 32, no. 2, pp. 374–382, 1985.
[11] P. Dutta, R. Guerraoui, and M. Vukolic, “Best-case complexity of asynchronous byzantine consensus,” Tech. Rep. EPFL/IC/2004/99, 2005.
[12] J. Sousa and A. Bessani, “From byzantine consensus to bft state machine replication: A latency-optimal transformation,” in proc. Dependable Computing Conf. IEEE, 2012, pp. 37–48.
[13] L. Su and N. H. Vaidya, “Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms,” in Proc. 2016 ACM Symposium on Principles of Distributed Computing, 2015.
[14] A. Nedić and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” IEEE Trans. Autom. Control, vol. 54, no. 1, pp. 48–61, 2009.
[15] R. S. S. A. Nedić, and V. V. Veeravalli, “Distributed stochastic subgradient projection algorithms for convex optimization,” J. Optimiz. Theory Appliat., vol. 147, no. 3, pp. 516–545, 2010.
[16] A. Nedić and A. Ozdaglar, “Distributed optimization over time-varying directed graphs,” IEEE Trans. Autom. Control, vol. 60, no. 3, pp. 601–615, 2015.
[17] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, “D-ADMM: A communication-efficient distributed algorithm for separable optimization,” IEEE Trans. Sig. Process., vol. 61, no. 10, pp. 2718–2723, 2013.
[18] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, “On the linear convergence of the ADMM in decentralized consensus optimization,” IEEE Trans. Sig. Process., vol. 62, no. 7, pp. 1750–1761, 2014.
[19] P. A. Forero, A. Cano, and G. B. Giannakis, “Consensus-based distributed support vector machines,” J. Mach. Learn. Res., vol. 11, pp. 1663–1707, 2010.
[20] Y. M. Minsky and F. B. Schneider, “Tolerating malicious gossip,” Distributed Computing, vol. 16, no. 1, pp. 49–68, 2003.
[21] H. J. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram, “Resilient asymptotic consensus in robust networks,” IEEE J. Sel. Areas Commun., vol. 31, no. 4, pp. 766–781, 2013.
[22] Z. Yang and W. U. Bajwa, “RD-SVM: A resilient distributed support vector machine,” in 2016 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2444–2448.
[23] Y. Chen, L. Su, and J. Xu, “Distributed statistical machine learning in adversarial settings: Byzantine gradient descent,” arXiv preprint arXiv:1705.05491, 2017.
[24] N. H. Vaidya, L. Tseng, and G. Liang, “Iterative approximate byzantine consensus in arbitrary directed graphs,” in Proc. 2016 ACM Symposium on Principles of Distributed Computing, 2012, pp. 365–374.
[25] N. H. Vaidya and V. K. Garg, “Byzantine vector consensus in complete graphs,” in Proc. 2016 ACM Symposium on Principles of Distributed Computing, 2013, pp. 65–73.
[26] N. H. Vaidya, “Iterative byzantine vector consensus in incomplete graphs,” Int. Conf. Distributed Computing and Networking, pp. 14–28, 2014.
[27] V. Vapnik, The nature of statistical learning theory. Springer science & business media, 2013.
[28] T. Do and F. Poulet, “Classifying one billion data with a new distributed svm algorithm,” in Proc. Int. Conf. Research Innovation and Vision for the Future, 2006, pp. 59–66.
[29] A. Nasia-Vázquez and E. Parrado-Hernandez, “Distributed support vector machines,” IEEE Trans. Neural Netw., vol. 17, no. 4, pp. 1091–1097, 2006.
[30] L. Su and N. Vaidya, “Byzantine multi-agent optimization: Part I,” arXiv preprint arXiv:1506.04681, 2015.
[31] N. Vaidya, “Matrix representation of iterative approximate byzantine consensus in directed graphs,” arXiv preprint arXiv:1203.1888, 2012.
[32] S. J. Wright, “Coordinate descent algorithms,” Mathematical Programming, vol. 151, no. 1, pp. 3–34, 2015.
[33] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” J. American statistical association, vol. 58, no. 301, pp. 13–30, 1963.
[34] J. Verger-Gaugry, “Covering a ball with smaller equal balls in $\mathbb{R}^n$,” Discrete & Computational Geometry, vol. 33, no. 1, pp. 143–155, 2005.