BYRDIE: A BYZANTINE-RESILIENT DISTRIBUTED LEARNING ALGORITHM

Zhixiong Yang and Waheed U. Bajwa

Department of Electrical and Computer Engineering,
Rutgers University–New Brunswick, Piscataway, NJ 08854
{zhixiong.yang, waheed.bajwa}@rutgers.edu

ABSTRACT

In this paper, a *Byzantine-resilient distributed coordinate descent* (ByRDiE) algorithm is introduced to accomplish machine learning tasks in a fully distributed fashion when there are Byzantine failures in the network. When data is distributed over a network, it is sometimes desirable to implement a fully distributed learning algorithm that does not require sharing of raw data among the network entities. To this end, existing distributed algorithms usually count on the cooperation of all nodes in the network. However, real-world applications often encounter situations where some nodes are either not reliable or are malicious. Such situations, in which some nodes do not behave as intended, can be modeled as having undergone Byzantine failures. Generally, Byzantine failures are hard to detect and can lead to break down of distributed learning algorithms. In this paper, it is shown that ByRDiE can provably tolerate Byzantine failures in the network under certain assumptions on the network topology and the machine learning tasks. ByRDiE accomplishes this by incorporating a local "screening" step into the update of a distributed coordinate descent algorithm. Finally, numerical results reported in the paper confirm the robustness of ByRDiE to Byzantine failures.

***Index Terms*—** Byzantine failure, distributed optimization, empirical risk minimization, machine learning, multiagent networks

1. INTRODUCTION

In machine learning, learning of models require the use of (labeled or unlabeled) training data. Traditionally, training data have been assumed available at a centralized location. In some recent applications, such as the internet-of-things, multiagent networks, and large-scale machine learning, training data tend to be distributed across different locations. Training of machine learning models in this setting of distributed datasets is often referred to as distributed learning [1, 2].

While distributed learning has received a lot of attention in recent years, most existing works make a simplified assumption that all nodes in the network operate as expected. Unfortunately, this assumption does not always hold true in practice; examples include cyber attacks, malfunctioning equipments and undetected failures [3,4]. When a node arbitrarily deviates from its intended behavior, it is termed to have undergone Byzantine failure [5]. While Byzantine failures are hard to detect in general, they can easily jeopardize the operation of the whole network [6–8].

In particular, with just a simple strategy, one can show that a single Byzantine node in the network can lead to failures of most state-of-the-art distributed learning algorithms [22]. The main contribution of this paper is to introduce and analyze an algorithm that completes distributed learning in the presence of Byzantine failures.

1.1. Related work and our contributions

A distributed learning problem can often be modeled as a distributed optimization problem by defining and minimizing a loss function on the training data of each node. Several types of distributed optimization algorithms have been introduced in the past to solve this problem. These include gradient-based methods [10–12], augmented Lagrangian-based methods [13–15] and second-order methods [16, 17]. While any of these algorithms can be used to solve a distributed learning problem, they all make the idealistic assumption that there are no failures in the network.

Byzantine-resilient algorithms have been studied extensively over the years [5, 18]. Byzantine-resilient algorithms for scalar averaging distributed consensus were studied in [19]. The algorithms proposed in [9, 22] extend this work from scalar consensus to scalar-valued distributed optimization, but they cannot be used in vector settings. The work in [24] introduces a method to implement distributed support vector machine (SVM) under Byzantine failures but the method does not generalize to other learning problems. A recent work [20] solves a vector-valued distributed learning problem under Byzantine failures, but it requires a central processing center. Because of this, it is not applicable in fully distributed settings.

There are several limitations of works like [9, 19–24]. One of the limitations is that the proposed algorithms pursue the minimizer of a convex combination of local empirical risk functions. This minimizer is usually different from the minimizer of the exact average of local loss functions. As such, there are no guarantees that the outputs of these algorithms result in either the minimum empirical risk or the minimum statistical risk. Another limitation is that, when forced to work with vectors, existing Byzantine-resilient algorithms require a strong assumption on the network topology [25]. Specifically, the smallest size of neighborhood of each node in the vector setting depends linearly on the dimensionality of the problem. This is impractical for most learning problems since the dimensionality of the training samples is usually much larger than the size of the neighborhood of each node.

Our work has two main contributions. First, we propose a coordinate descent-based distributed algorithm that can solve a high-dimensional distributed learning problem under mild assumptions on the network topology and the (regularized) loss function. To the best of our knowledge, this problem had been unsolved prior to this work. It is worth noting here that the semi-distributed Byzantine-resilient learning problem studied in works such as [20] is very different from the fully distributed setting of this work. Byzantine nodes are much more powerful and dangerous when there is no central processor in

This work is supported in part by the NSF under award CCF-1453073, by the ARO under award W911NF-17-1-0546, and by the DARPA Lagrange Program under ONR/SPAWAR contract N660011824020.

the network that can coordinate different steps of the learning algorithm. Second, while existing works can only achieve the minimum of a convex combination of local empirical risk functions, we provide theoretical guarantees that the output of our algorithm converges to the minimum of the statistical risk under the assumption of independent and identically distributed (i.i.d.) training samples.

1.2. Notation and organization

All vectors are taken to be column vectors. We use $[a]_k$ and $[A]_{ij}$ to denote the k-th element of vector a and the (i,j)-th element of matrix A, respectively. We use $\|a\|$ to denote ℓ_2-norm of a and $\|a\|_\infty$ to denote its ℓ_∞-norm. Given a set, $|\cdot|$ denotes its cardinality, while $(\cdot)^T$ denotes the transpose operation. Finally, 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 follows. Section 2 gives the problem formulation. Section 3 discusses the ByRDiE algorithm along with the theoretical guarantees. Numerical results are provided in Section 4. Section 5 concludes the paper.

2. PROBLEM FORMULATION

Given a network in which each node has access to some local training data, the main goal of this paper is to learn a machine learning model from the data in a distributed fashion, even in the presence of Byzantine failures.

2.1. Distributed learning model

We consider a network of M nodes, expressed as a directed, static graph $\mathcal{G}(J,\mathcal{E})$. Here, the set $J := \{1,\ldots,M\}$ represents nodes in the network, while the set of edges $\mathcal{E}$ represents communication links between different nodes. Specifically, $(j,i) \in \mathcal{E}$ 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, $y \in \mathbb{R}$.[1] 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 cardinalities of local training sets are the same, i.e., $|S_j| = N$. The generalization to the case when S_j's are not equal is trivial.

The goal in distributed learning is to learn a function using the distributed training data that correctly maps x to y. One popular mapping is $y = w^T x$. To find a "good" w, one first defines a loss function $\ell(w,(x,y))$, where the value of loss function increases when the difference between the mapping of x and y increases. To avoid over fitting, a regularizer $R(w)$ is often added to the loss function. Then one can solve for w by statistically minimizing a regularized loss function $f(w,(x,y)) \triangleq R(w) + \ell(w,(x,y))$. The regularized $f(w,(x,y))$ is often referred to as risk function. In this paper, we focus on the class of convex differentiable loss functions and strongly convex and smooth regularizers[2]. Examples include square loss $(1 - y \cdot w^T x)^2$, square hinge loss $\max(0, 1 - y \cdot w^T x)^2$, logistic loss $\ln(1+e^{-y \cdot w^T x})$ and $R(w) = \frac{\lambda}{2}\|w\|^2$. In this paper, we make the following Lipschitz continuity assumption on the gradient of $f(\cdot)$.

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

Note that Assumption 1 implies that the risk function itself is also Lipschitz, i.e., $\|f(w_1,(x,y)) - f(w_2,(x,y))\| \leq L'\|w_1 - w_2\|$ [30]. Ideally, we need to learn the "best" mapping as follows:

$$w^* = \arg\min_{w \in W} \mathbb{E}_{(x,y)\sim D}[f(w,(x,y))]. \quad (1)$$

Here, $W \subset \mathbb{R}^P$ denote the feasible set for w. In this work, we take W to be the closed and compact set $\{w : \|w\|_\infty \leq \Gamma\}$. Note that solving an unconstrained problem is equivalent to solving (1) on the constraint set W with Γ large. Extensions to other convex constraint sets can be carried out with slight modifications to our main algorithm, but would not be pursued in this work. Next, we make another assumption.

Assumption 2. *For any $w \in W$, the loss function $\ell(w,(x,y))$ is bounded almost surely over all training samples, i.e., $\ell(w,(x,y)) \leq C < \infty$, $\forall (x,y) \in \bigcup_{j\in J} S_j$.*

Note that Assumption 2 would be satisfied for datasets with finite-valued training samples because of the Lipschitz continuity of $\ell(w,(x,y))$ and the compactness of W.

Since D is unknown, we can not solve for w^* directly. In many distributed learning problems, a broadly adopted alternative is to define and minimize the average of all local empirical risk functions, i.e., $\frac{1}{M}\sum_{j=1}^{M} \widehat{f}(w,S_j) \triangleq R(w) + \frac{1}{MN}\sum_{j=1}^{M}\sum_{n=1}^{N} \ell(w,(x_{jn},y_{jn}))$. To achieve this goal, we need nodes to cooperate with each other by communicating over edges. Specifically, define the neighbourhood of j as $\mathcal{N}_j := \{i \in J : (i,j) \in \mathcal{E}\}$. We say that node i is a neighbour of node j if $i \in \mathcal{N}_j$. Distributed learning algorithms proceed iteratively. In each iteration of the algorithm, node j is expected to accomplish two tasks: (1) update a local variable w_j according to some rule $g_j(\cdot)$, and (2) broadcast the updated local variable to other nodes, where node i can receive the broadcasted information from node j *only* if $j \in \mathcal{N}_i$.

2.2. Byzantine failure model

While distributed learning via message passing is well-understood [15, 26, 27], existing 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 failures, formally defined as follows.

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

In this paper, we assume there are at most b Byzantine nodes in the network. Let $J' \subset J$ denote the set of non-faulty nodes. Without loss of generality, we assume that non-faulty nodes are labeled from 1 to $|J'|$. We now provide some definitions and assumptions that are common in the literature; see, e.g., , [22].

Definition 2. *A subgraph $\mathcal{G}'$ of $\mathcal{G}$ is called a reduced graph if it is generated from graph $\mathcal{G}$ 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.*

[1] Note that while the main problem is being formulated here under the supervised setting, our proposed framework and the final results are equally applicable under both unsupervised and semi-supervised settings.

[2] A function $R(w)$ is strongly convex if it satisfies $R(zw_1+(1-z)w_2) < zR(w_1) + (1-z)R(w_2)$ for any $0 < z < 1$. Further, $R(w)$ is smooth if it is differentiable for all orders.

Definition 3. *A "source component" of graph $\mathcal{G}'$ is a collection of nodes such that each node in the source component has a directed path to every other node in $\mathcal{G}'$.*

Assumption 3. *All reduced graphs $\mathcal{G}'$ generated from $\mathcal{G}(J, \mathcal{E})$ contain a source component of cardinality at least $b + 1$.*

Assumption 3 is to ensure that there is enough redundancy in the graph to tolerate Byzantine failures. Note that the total number of different reduced graphs one can generate from $\mathcal{G}$ is finite as long as M is finite. So, in theory, the assumption can be checked for any graph with a finite number of nodes. However, checking this assumption efficiently still remains an open question. In the case of Erdös–Rényi graphs used in our experiments, however, we have observed that Assumption 3 is typically satisfied whenever the ratio of the average incoming degree of the graph and the number of Byzantine nodes is high enough. Note that it is not necessary for our algorithm to know the exact value of b. For example, we can set b as an upper bound on the number of Byzantine nodes. As long as the network topology assumption is satisfied, our proposed algorithm can can tolerate up to b Byzantine nodes.

The goal of this paper is to develop a Byzantine fault-tolerant algorithm for distributed learning under Assumptions 1–3. In particular, under the assumption of at most b Byzantine nodes in the network, we need to accomplish the following: (i) achieve consensus among nonfaulty nodes, i.e., $w_j^r = w_i^r \ \forall i, j \in J'$ as the number of iterations $r \to \infty$; (ii) ensure that $w_j^r \to w^* \ \forall j \in J'$ as the sample size $N \to \infty$.

3. BYRDIE: BYZANTINE-RESILIENT DISTRIBUTED COORDINATE DESCENT

Our goal is to find the global optimum of the problem

$$w_{opt} = \arg\min_{w \in W} R(w) + \frac{1}{MN} \sum_{j=1}^{M} \sum_{n=1}^{N} \ell(w, (x_{jn}, y_{jn})) \qquad (2)$$

at each node $j \in J'$ and show that $w_{opt} \to w^*$ as $N \to \infty$. We already know from prior work [28] that the exact optimum of (2) is not achievable when $b \geq 1$. As an alternative, existing works [21–23,25] pursue a convex combination of local empirical risks. In contrast, we introduce an algorithm called **By**zantine-**R**esilient **Di**stributed coordinate d**E**scent (ByRDiE) and show that the algorithm achieves the statistical optimum at each node $j \in J'$ in the presence of Byzantine failures when the training data is i.i.d..

The idea behind ByRDiE is to convert a vector-valued optimization problem into a sequence of scalar-valued subproblems at each dimension using coordinate descent. Afterward, it performs a Byzantine-resilient exact line search in each dimension. The process of ByRDiE is shown in Algorithm 1. This implementation can be broken into a coordinate descent loop (step 2) and an inner loop (step 4). The outer loop is the coordinate descent loop that breaks the vector optimization problem into P scalar-valued subproblems. The inner loop solves each subproblem and ensures resilience to Byzantine failures.

We assume that the total number of iterations $\bar{r}$ for coordinate descent is specified during the initialization. The number of iterations $T(\bar{r})$ for each scalar-valued subproblem within coordinate descent is assumed to be a function of $\bar{r}$ that satisfies $T(\bar{r}) \to \infty$ as $\bar{r} \to \infty$. We use $[w_j^r(t)]_k$ to denote the k-th element of w_j at the r-th iteration of the coordinate descent loop and the t-th iteration of the k-th subproblem (coordinate). Without loss of generality, we initialize $[w_j^1]_k = 0, \forall k = 1, ..., P$.

Algorithm 1 ByRDiE for distributed learning

Input: $S_1, S_2, \ldots, S_M, b, \{\rho(t)\}_{t=1}^{\infty}, \bar{r}, T(\bar{r})$
1: **Initialize** $[w_j^1(1)]_k \leftarrow 0$, $j = 1, \ldots, |J'|$, $k = 1, \ldots, P$
2: **for** $r = 1, 2, 3, \ldots, \bar{r}$ **do**
3: **for** $k = 1, 2, 3, \ldots, P$ **do**
4: **for** $t = 1, 2, 3, \ldots, T(\bar{r})$ **do**
5: **for** $j = 1, 2, 3, \ldots, |J'|$ **do In parallel**
6: Receive $[w_i^r(t)]_k$ from all $i \in \mathcal{N}_j$
7: Find $\mathcal{N}_j^s(r, k, t)$, $\mathcal{N}_j^l(r, k, t)$, $\mathcal{N}_j^*(r, k, t)$ according to (3), (4) and (5).
8: Update $[w_j^r(t+1)]_k$ as in (6)
9: **end for**
10: **end for**
11: $[w_j^{r+1}(1)]_k \leftarrow [w_j^r(T(\bar{r}))]_k \ \forall j \in J', k = 1, \ldots, P$
12: **end for**
13: **end for**
Output: $\{w_j^{\bar{r}}(T(\bar{r}))\}_{j \in J'}$

At the beginning of each inner loop (for some r and k), every node initializes from $[w_j^r(1)]_k$. Then during each iteration t, every node engages in the following: broadcast, screening and update. In the broadcast step, node j receives $[w_i^r(t)]_k$ from all neighbors $i \in \mathcal{N}_j$. During this step, a node can receive values from both Byzantine and non-faulty neighbors. The main idea of screening 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 thus break $\mathcal{N}_j$ into 3 subsets $\mathcal{N}_j^*(r, k, t)$, $\mathcal{N}_j^s(r, k, t)$ and $\mathcal{N}_j^l(r, k, t)$, which are defined as the following:

$$\mathcal{N}_j^s(r, k, t) = \arg\min_{X : X \subset \mathcal{N}_j, |X| = b} \sum_{i \in X} [w_i^r(t)]_k, \qquad (3)$$

$$\mathcal{N}_j^l(r, k, t) = \arg\max_{X : X \subset \mathcal{N}_j, |X| = b} \sum_{i \in X} [w_i^r(t)]_k, \qquad (4)$$

$$\mathcal{N}_j^*(r, k, t) = \mathcal{N}_j \setminus \mathcal{N}_j^s(r, k, t) \setminus \mathcal{N}_j^l(r, k, t). \qquad (5)$$

One way of finding the three sets is by a sorting process. Node j sorts $[w_i^r(t)]_k$'s in an increasing order for all $i \in \mathcal{N}_j$, breaking ties arbitrarily. Then node j adds the neighbors that broadcast the b largest $[w_i^r(t)]_k$'s into $\mathcal{N}_j^l(t)$. Similarly, node j can evaluate $\mathcal{N}_j^s(r, k, t)$. Then it adds all the neighbors that are not in $\mathcal{N}_j^l(r, k, t)$ or $\mathcal{N}_j^s(r, k, t)$ into $\mathcal{N}_j^*(r, k, t)$. The step is called screening because node j only selects $[w_i^r(t)]_k$'s from $\mathcal{N}_j^*(r, k, t)$ to update its local value. Note that there might still be $[w_i^r(t)]_k$'s received from Byzantine nodes in $\mathcal{N}_j^*(r, k, t)$. With a slight abuse of notation, we use $[\nabla \widehat{f}(w_j^r(t), S_j)]_k$ to denote the k-th element of $\nabla \widehat{f}$ at the t-th iteration of the k-th inner loop and the r-th iteration of the outer loop. Finally, the update rule at node j can be described as follows:

$$[w_j^r(t+1)]_k = \frac{1}{|\mathcal{N}_j| - 2b + 1} \sum_{i \in \mathcal{N}_j^*(r,k,t) \cup \{j\}} [w_i^r(t)]_k - \rho(t)[\nabla \widehat{f}(w_j^r(t), S_j)]_k, \qquad (6)$$

where $\{\rho(t)\}_{t=1}^{\infty}$ is a sequence of stepsizes satisfying $0 < \rho(t+1) \leq \rho(t)$, $\sum_{t=1}^{\infty} \rho(t) = \infty$ and $\sum_{t=1}^{\infty} \rho^2(t) < \infty$. Note that Assumption 3 is required to ensure that the screening step within the inner loop can reliably thwart the Byzantine nodes [22]. The next inner loop starts with the output of the previous inner loop. Each coordinate

descent iteration r starts with the first dimension and ends after the P-th dimension is processed, which is a standard coordinate descent process. We now state our main result.

Theorem 1. *Let Assumptions 1, 2 and 3 hold. Then all non-faulty nodes in the network achieve consensus, i.e., $\exists w^{\infty}$ such that $\forall j \in J'$, $w_j^{\bar{r}}(T(\bar{r})) \xrightarrow{\bar{r}} w^{\infty}$. Further, we also have*

$$\mathbb{E}_{(x,y)\sim D}[f(w^{\infty},(x,y))] \xrightarrow{N} \mathbb{E}_{(x,y)\sim D}[f(w^{*},(x,y))] \quad (7)$$

in probability.

Note that both the empirical risk and the statistical risk functions are strongly convex. Therefore, Theorem 1 can also be interpreted as $w_j^{\bar{r}}(T(\bar{r})) \to w^*$ due to the uniqueness of the minimum of strongly convex functions. The rigorous proof of this theorem can be found in the extended version of this work [29]. Here we heuristically describe why the algorithm works. First, the output of each inner loop can be shown to be the minimum of a convex combination of all local loss functions with respect to one dimension k as $T(\bar{r}) \to \infty$. Then, given the data samples are i.i.d., this minimum converges to the minimum of the statistical risk with respect to the k-th dimension in probability as a function of N. Thus as k increases from 1 to P and r increases from 1 to $\bar{r}$, the outer loop can be viewed as repeating the following steps: (1) minimize the statistical risk along dimension k, and (2) switch to a different k. This process is a well-understood coordinate descent process, which can be shown to converge to the minimum of the statistical risk as $\bar{r} \to \infty$.

4. NUMERICAL RESULTS

In this section, we use our algorithm for classificaion of the MNIST8M handwritten digits dataset [31] distributed over a network of 800 nodes to show two facts: (i) the classic distributed learning algorithms fail when there are Byzantine failures in the network, and (ii) ByRDiE is Byzantine resilient and its performance is comparable to the case when there are no Byzantine failures in the network. We use a linear support vector machine (SVM) to train the classifier. The dimensionality of each image in MNIST8M is 784 and the network has 800 nodes, so the network does not satisfy the strong constraint of [23]. In particular, there is no existing distributed method that solves this classification problem in the presence of Byzantine failures to the best of our knowledge. So we use the accuracy of centralized SVM as the baseline. We perform training using ByRDiE and compare our results with both the baseline and distributed gradient descent (DGD) [10]. We perform four sets of experiments: (1) Collect all training samples together and run centralized SVM; (2) SVM via DGD with Byzantine failures in the network; (3) SVM via ByRDiE with Byzantine failures in the network; and (4) Centralized SVM using the local data on each node.

We model the problem as a simple binary classification problem of distinguishing between digits '5' and '8', which are the two most inseparable digits. We distribute training samples onto 800 nodes with 250 digit '5' and 250 digit '8' per node. The test set is of size 40,000 with an equal number of both digits. We use an Erdös–Rényi graph with 800 nodes and probability of being connected to be 0.5. We randomly pick 20 nodes to be Byzantine nodes, each of which broadcasts random vectors to its neighbors in each iteration.

Two metrics should be taken into account for these experiments: accuracy and consensus. Accuracy is measured by the rate of classifying test samples correctly. We model the accuracy and consensus metrics as functions of outer loop iterations, i.e., one round of updates from the first dimension to the last. If the ℓ_2-norm of the difference between any pair of classifiers is less than 1% of the ℓ_2-norm of both classifiers, we say that the algorithm has achieved consensus.

Table 1. MNIST8M dataset test

Algorithm	Sample size	Accuracy	Consensus
Centralized SVM	400000×1	92.30%	N/A
SVM via DGD	500×800	50%	NO
SVM via ByRDiE	500×800	91.83%	YES
Local SVM	500×1	88.08%	N/A

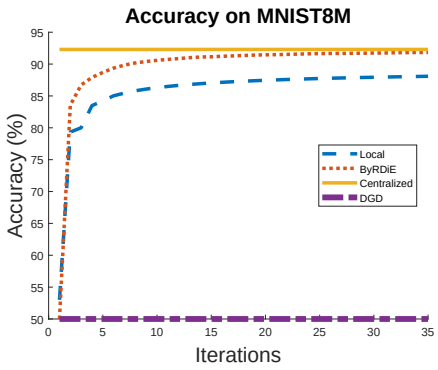

Fig. 1. The average accuracy of all local classifiers on the test dataset is used as the metric of performance of algorithms. The solid line is the accuracy of the centralized method with all training samples, which is the baseline of performance. The accuracy of ByRDiE is close to the baseline while DGD fails because of the Byzantine failures. The gap between ByRDiE and training using local data shows that by taking advantage of cooperation among all nodes, ByRDiE can maintain a competitive performance even when there are Byzantine failures in the network.

The results are shown in Table 1 and Fig. 1. As a baseline, centralized SVM achieves the highest accuracy of 92.30%. Classic DGD has an accuracy of 50% since it gives all test data the same label. In addition, consensus is not achieved by DGD 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, when a node chooses not to cooperate with others and performs training only on local data, it achieves an accuracy of 88.08%. In contrast, ByRDiE achieves consensus on a classifier that has 91.83% accuracy, which shows that ByRDiE can indeed maintain a competitive performance by cooperation among all nodes while staying resilient to Byzantine failures.

5. CONCLUSION

In this paper, we introduced a Byzantine-resilient distributed coordinate descent algorithm (ByRDiE). The algorithm minimizes the statistical risk in a fully distributed fashion. Numerical results show that when there are Byzantine failures in the network, ByRDiE can deliver a competitive performance while existing algorithms fail. In terms of future work, the technique that helps ByRDiE reject Byzantine failures can potentially be combined with other optimization methods to increase convergence rate, reduce communication cost or lower data complexity requirements.

6. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[6] 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.

[7] P. Dutta, R. Guerraoui, and M. Vukolic, "Best-case complexity of asynchronous Byzantine consensus," *Tech. Rep. EPFL/IC/200499*, 2005.

[8] 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.

[9] L. Su and N. H. Vaidya, "Fault-tolerant multi-agent optimization: Optimal iterative distributed algorithms," in *Proc. ACM Symposium on Principles of Distributed Computing*, pp.424–434, 2016.

[10] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, 2009.

[11] S.S. Ram, A. Nedić, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *J. Optimiz. Theory Applicat.*, vol. 147, no. 3, pp. 516–545, 2010.

[12] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 601–615, 2015.

[13] 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.

[14] 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.

[15] 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.

[16] P. A. Mokhtari, Q. Ling, and A. Ribeiro, "Network newton distributed optimization methods," *IEEE Trans. Sig. Process.*, vol. 65, no. 1, pp. 146–161, 2017.

[17] P. A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, "A decentralized second-Order method with exact linear convergence rate for consensus optimization," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 4, pp. 507–522, 2016.

[18] Y. M. Minsky and F. B. Schneider, "Tolerating malicious gossip," *Distributed Computing*, vol. 16, no. 1, pp. 49–68, 2003.

[19] 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.

[20] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *arXiv preprint arXiv:1705.05491*, 2017.

[21] N. H. Vaidya, L. Tseng, and G. Liang, "Iterative approximate Byzantine consensus in arbitrary directed graphs," in *Proc. ACM Symposium on Principles of Distributed Computing*, 2012, pp. 365–374.

[22] L. Su and N. Vaidya, "Fault-tolerant distributed optimization (Part IV): Constrained optimization with arbitrary directed networks," *arXiv preprint arXiv:1511.01821*, 2015.

[23] N. H. Vaidya, "Iterative Byzantine vector consensus in incomplete graphs," in *Proc. Int. Conf. Distributed Computing and Networking*, pp. 14–28, 2014.

[24] Z. Yang and W. U. Bajwa, "RD-SVM: A resilient distributed support vector machine," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2444–2448.

[25] N. H. Vaidya and V. K. Garg, "Byzantine vector consensus in complete graphs," in *Proc. ACM Symposium on Principles of Distributed Computing*, 2013, pp. 65–73.

[26] 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.

[27] A. Navia-Vázquez and E. Parrado-Hernandez, "Distributed support vector machines," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 1091–1097, 2006.

[28] L. Su and N. Vaidya, "Byzantine multi-agent optimization: Part I," *arXiv preprint arXiv:1506.04681*, 2015.

[29] Z. Yang and W.U.Bajwa, "ByRDiE: Byzantine-resilient distributed coordinate descent for decentralized learning," *arXiv preprint arXiv:1708.08155*, 2017.

[30] H. H.Sohrab, *Basic real analysis*. Birkhauser Boston, 2003.

[31] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," *Large scale kernel machines*, pp. 031–320, 2007.