PAC LEARNING FROM DISTRIBUTED DATA IN THE PRESENCE OF MALICIOUS NODES

Zhixiong Yang and Waheed U. Bajwa

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

ABSTRACT

When data is distributed over a network, statistical learning needs to be carried out in a fully distributed fashion. When all nodes in the network are faultless and cooperate with each other, it is understood that the objective is *probably approximately correct* (PAC) learnable. However, when there are malicious nodes trying to sabotage learning by injecting false information in the network, PAC learnability of the objective remains an open question. In this paper, we discuss the distributed statistical learning problem when the risk function is strictly convex. We show that the model is PAC learnable in the presence of malicious nodes by proposing and analyzing a distributed learning algorithm. Experiments in non-convex settings are also performed to further discuss the PAC learnability of non-convex statistical learning problems from distributed data.

1. INTRODUCTION

Machine learning (ML) tasks involve statistically minimizing a risk function $f(\mathbf{w}, \mathbf{z})$ with respect to the model variable $\mathbf{w}$, where $\mathbf{z}$ denotes random data drawn from some unknown distribution $\mathcal{D}$. While ML problems are traditionally solved by empirical risk minimization (ERM) techniques, the problems can be fit into a more general *probably approximately correct* (PAC) learning framework [1, 2]. The understanding of PAC learnability of a ML problem gives more insights into its generalization capabilities over unseen data.

When the training data is available at a single location, it is understood that the hypothesis/model $\mathbf{w}$ is PAC learnable [2]. In the modern world, the scenarios in which the datasets are distributed over a network frequently appear due to reasons such as storage and computation constraints, privacy concerns, and engineering needs. Learning tasks in such cases have to be completed by nodes cooperating with each other over the network, with numerous distributed learning algorithms developed over the years to solve the resulting distributed problems [3–5]. It can also be shown that the hypothesis $\mathbf{w}$ is PAC learnable in the distributed setting. However, unlike the centralized setting, it is more often than not that not all the nodes in the network can be trusted. Indeed, nodes may undergo undetected failures or even become malicious in the distributed setting. One consequence of this is that the aforementioned PAC learnability analysis is no longer valid in a malicious environment.

In this paper, we discuss the PAC learnability under distributed settings in the presence of malicious nodes. Specifically, we consider a graph (network) of M nodes of which at most b nodes are malicious. Each node can independently perform some computations on its local dataset and the nodes can communicate with each other over graph edges. Our goal is to show the PAC learnability of the hypothesis $\mathbf{w}$ in this setting when the risk function is strictly convex with respect to $\mathbf{w}$ and has Lipschitz gradients.

Related work. The PAC learnability of machine learning models in centralized settings has been well understood in the last few decades [1, 2]. With the increasing need for fully distributed learning, researchers have been developing distributed learning algorithms [3–5]. It has in particular been shown that, when data is distributed over a network, the class of machine learning hypotheses is also PAC learnable over a faultless network [6, 7]. Recently, researchers have looked for ways to improve the robustness of distributed learning against malicious attacks. In this regard, several robust optimization algorithms have been developed to counter the presence of potential malicious nodes in the network [8–13]. Nonetheless, the topic of PAC learnability in the presence of malicious nodes remains relatively open.

Our contribution. One of the classic ways to accomplish ML tasks is via the ERM framework. Intuitively, centralized ERM finds the minimizer of the empirical risk and the empirical minimizer can then be shown to converge to the minimizer of the statistical risk function. It can then be deduced from such convergence analysis that the class of ML hypotheses is PAC learnable. Distributed ERM finds the minimizer of the distributed empirical risk function. When the distributed dataset is created by distributing the centralized dataset over a network, it can be shown that the solution of the distributed ERM problem is equivalent to the solution of the centralized ERM problem. Consequently, taking advantage of the analysis for centralized ERM, it can be concluded that the class of ML hypotheses is also PAC learnable from data distributed over a faultless network.

Unfortunately, it has been shown in previous work [14] that the distributed ERM problem cannot be exactly solved when there is even one malicious node in the network. The infeasibility of the distributed empirical minimizer leads to the consequence that the convergence analysis from distributed ERM to centralized ERM and then to the true (i.e., statistical) minimizer is no longer applicable under the malicious settings; this then makes the PAC learnability questionable. In this paper, we establish for the first time in the literature that, for a limited number of malicious nodes, the true hypothesis is PAC learnable in the presence of malicious nodes when the risk function is strictly convex. We provide a distributed learning algorithm that is easy to implement and show the PAC learnability based on the given algorithm. As we will show later, the algorithm and its analysis—as opposed to the classic distributed ERM framework—do not involve exactly finding the empirical minimizer.

Paper organization. The rest of the paper is organized as the following. The problem is formulated in Section 2. We present our algorithm and analysis in Section 3. Numerical results are given in Section 4 and we conclude the paper in Section 5.

This work is supported in part by the NSF under awards CCF-1453073 and CCF-1907658, by the ARO under award W911NF-17-1-0546, and by the DARPA Lagrange Program under ONR/NIWC contract N660011824020.

2. PROBLEM FORMULATION

Consider a set of independent and identically distributed (i.i.d.) samples of random variables $\mathbf{z}$ that follow distribution $\mathcal{D}$, which are separated into M datasets, each of size N. The datasets are distributed among M nodes over a network. The network is expressed as a directed, static graph $\mathcal{G}(\mathcal{J}, \mathcal{E})$. Here, the set $\mathcal{J} := \{1, \ldots, M\}$ 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 messages from node j. We denote the dataset at node j as $\mathbf{Z}_j$ of which the n-th sample is denoted as $\mathbf{z}_{jn}$.

Let the class of hypotheses be $\{\mathbf{w} \in \mathbb{R}^P\}$ and consider a risk function $f(\mathbf{w}, \mathbf{z})$ that describes the correctness of any hypothesis $\mathbf{w}$. In this paper, we focus on the class of strictly convex risk functions with Lipschitz gradients, which is formally stated as the following.

Assumption 1. The risk function $f(\mathbf{w}, \mathbf{z})$ satisfies:

1. $\forall \mathbf{w}_1, \mathbf{w}_2, \forall a \in (0, 1)$, $f(a\mathbf{w}_1 + (1-a)\mathbf{w}_2, \mathbf{z}) < af(\mathbf{w}_1, \mathbf{z}) + (1-a)f(\mathbf{w}_2, \mathbf{z})$;
2. $\forall \mathbf{w}_1, \mathbf{w}_2, \|f(\mathbf{w}_1, \mathbf{z}) - f(\mathbf{w}_2, \mathbf{z})\|_2 \leq L\|\mathbf{w}_1 - \mathbf{w}_2\|_2$;
3. $\forall \mathbf{w}_1, \mathbf{w}_2, \|\nabla f(\mathbf{w}_1, \mathbf{z}) - \nabla f(\mathbf{w}_2, \mathbf{z})\|_2 \leq L'\|\mathbf{w}_1 - \mathbf{w}_2\|_2$.

Since we deal with finite values in the real world, we make another assumption on both the hypothesis $\mathbf{w}$ and the risk function.

Assumption 2. For any hypothesis $\mathbf{w}$ and any sample $\mathbf{z} \in \bigcup_{j \in J} \mathbf{Z}_j$, $f(\mathbf{w}, \mathbf{z})$ is bounded almost surely over all training samples: $0 \leq f(\mathbf{w}, \mathbf{z}) \leq C < \infty$.

We want to learn a good hypothesis $\mathbf{w}$ in a distributed fashion. To achieve this goal, we need nodes to cooperate with each other by communicating over network edges. Specifically, define the neighborhood of node j as $\mathcal{N}_j := \{i \in \mathcal{J} : (j, i) \in \mathcal{E}\}$. We say that node i is a neighbor of node j if $i \in \mathcal{N}_j$. Distributed learning algorithms proceed iteratively. In each iteration $(r+1)$ of the algorithm, node j is expected to accomplish two tasks:

1. Update a local variable $\mathbf{w}_j^r$ according to some (deterministic or stochastic) 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 $j \in \mathcal{N}_i$.

To discuss PAC learnability in the presence of malicious nodes, we assume there are at most b malicious nodes in the network. We model the behavior of malicious nodes as broadcasting arbitrary messages in each iteration (i.e., as *Byzantine* nodes [15]). Note that a malicious node can choose to follow the algorithm so that the exact number and identity of malicious nodes stay unknown. Let $\mathcal{J}' \subset \mathcal{J}$ denote the set of nonfaulty nodes and define $M' := |\mathcal{J}'|$. Without loss of generality, we assume the nonfaulty nodes are labeled from 1 to M'. Since distributed learning relies on message passing, it is obvious that if the presence of malicious nodes cuts the network into pieces, learning cannot take place. Therefore, we also need to assume that there is enough connectivity-based redundancy in the network so that it is possible to tolerate a certain number of malicious nodes. To this end, we provide dome definitions and an assumption that are common in the literature [8, 16, 17].

Definition 1. A subgraph $\mathcal{G}_r$ of $\mathcal{G}$ is called a reduced graph if it is generated by (i) removing all malicious nodes along with their incoming and outgoing edges, and (ii) removing additionally up to b incoming edges from each nonfaulty node.

Definition 2. A "source component" of a graph is a collection of nodes such that each node in the source component has a directed path to every other node in the graph.

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

In this paper, we will show that one can learn a hypothesis from distributed data in the presence of malicious nodes that is probably approximately correct. More specifically, under Assumptions 1, 2 and 3, we will show that one can learn a hypothesis $\bar{\mathbf{w}}$ at each nonfaulty node such that, for arbitrarily small ϵ and δ, $\mathbf{P}(\|\mathbb{E}[f(\bar{\mathbf{w}}, \mathbf{z})] - \mathbb{E}[f(\mathbf{w}^*, \mathbf{z})]\|_2 < \epsilon) \geq 1 - \delta$ can be achieved as a function of N, where $\mathbf{w}^*$ is the minimizer of the statistical risk, i.e., $\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^P} \mathbb{E}[f(\mathbf{w}, \mathbf{z})]$.

3. PAC LEARNING UNDER MALICIOUS SETTINGS

In this section, we first give our main result of PAC learnability from distributed data under malicious settings. Then we introduce a distributed learning algorithm and prove the result by showing that the output of the algorithm is probably approximately correct.

Theorem 1. *Let Assumptions 1, 2, and 3 be satisfied. There exists an algorithm that can learn a hypothesis $\bar{\mathbf{w}}$ at each nonfaulty node that is probably approximately correct. Specifically, with probability at least $1 - \mathcal{O}(\exp(-N\epsilon^2))$, $\|\mathbb{E}[f(\bar{\mathbf{w}}, \mathbf{z})] - \mathbb{E}[f(\mathbf{w}^*, \mathbf{z})]\|_2 < \epsilon$.*

3.1. A robust distributed learning algorithm

Since the malicious nodes try to sabotage the algorithm by sending false information, classic non-resilient distributed learning algorithms, e.g., distributed gradient descent (DGD) and distributed alternating direction method of multipliers (D-ADMM), cannot accomplish the learning task. We now introduce a robust learning algorithm termed *Byzantine-resilient distributed coordinate descent* (ByRDiE) and then we show that the output of ByRDiE is probably approximately correct.

ByRDiE involves splitting the distributed learning problem into a sequence of one-dimensional subproblems using coordinate descent and then approximately solving each scalar-valued subproblem using the approach described in [8]. The exact implementation is detailed in Algorithm 1. The algorithm can be broken into an outer loop (Step 2) and an inner loop (Step 4). The outer loop is the coordinate descent loop, which breaks the vector-valued optimization problem in each iteration r into P scalar-valued subproblems. The inner loop solves a scalar-valued optimization problem in each iteration t and ensures robustness against malicious behaviors. We assume the total number of iterations $\bar{r}$ for coordinate descent are specified during initialization. We use $[\mathbf{w}_j^r(t)]_k$ to denote the k-th element of $\mathbf{w}_j$ at the r-th iteration of the coordinate descent loop and the t-th iteration of the k-th inner loop. Without loss of generality, we initialize $[\mathbf{w}_j^1(1)]_k = 0, \forall k = 1, \ldots, P$.

We now fix some r and k, and focus on the implementation of the inner loop (Step 4). Every node has some $[\mathbf{w}_j^r(1)]_k$ at the start of the inner loop ($t = 1$). During each iteration t of this loop, all (nonfaulty) nodes engage in the following: broadcast, screening, and update. In the broadcast step (Step 6), all nodes $i \in \mathcal{J}$ broadcast $[\mathbf{w}_i^r(t)]_k$'s and each node $j \in \mathcal{J}$ receives $[\mathbf{w}_i^r(t)]_k, \forall i \in \mathcal{N}_j$. During this step, a node can receive values from both nonfaulty and malicious neighbors. The main idea of the screening step (Step 7) is to reject values at node j that are either "too large" or "too small" so that the values being used for update by node j in each iteration will

Algorithm 1 Byzantine-resilient distributed coordinate descent

Input: $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_M, \{\rho(\tau)\}_{\tau=1}^{\infty}, b \in \mathbb{N}, \bar{r} \in \mathbb{N}, T \in \mathbb{N}$

1: **Initialize:** $r \leftarrow 1$, $t \leftarrow 1$, and $\forall j \in \mathcal{J}'$, $\mathbf{w}_j^1(1) \leftarrow 0$
2: **for** $r = 1, 2, \ldots, \bar{r}$ **do**
3: **for** $k = 1, 2, \ldots, P$ **do**
4: **for** $t = 1, 2, \ldots, T$ **do**
5: **for** $j = 1, 2, \ldots, M'$ **do (in parallel)**
6: Receive $[\mathbf{w}_i^r(t)]_k$ 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 (1), (2), and (3)
8: Update $[\mathbf{w}_j^r(t+1)]_k$ as in (4)
9: **end for**
10: **end for**
11: **end for**
12: $\mathbf{w}_j^{r,T} \leftarrow \mathbf{w}_j^r(T+1), \forall j \in \mathcal{J}'$
13: $\mathbf{w}_j^{r+1}(1) \leftarrow \mathbf{w}_j^{r,T}, \forall j \in \mathcal{J}'$
14: **end for**

Output: $\left\{\mathbf{w}_j^{\bar{r},T}\right\}_{j\in\mathcal{J}'}$

be upper and lower bounded by a set of values generated by nonfaulty nodes. To this end, we partition $\mathcal{N}_j$ 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 following:

$$\mathcal{N}_j^s(r,k,t) = \underset{X:X\subset\mathcal{N}_j,|X|=b}{\arg\min} \sum_{i\in X} [w_i^r(t)]_k, \tag{1}$$

$$\mathcal{N}_j^l(r,k,t) = \underset{X:X\subset\mathcal{N}_j,|X|=b}{\arg\max} \sum_{i\in X} [w_i^r(t)]_k, \tag{2}$$

$$\mathcal{N}_j^*(r,k,t) = \mathcal{N}_j \setminus \mathcal{N}_j^s(r,k,t) \setminus \mathcal{N}_j^l(r,k,t). \tag{3}$$

The step is called screening because node j only uses $[\mathbf{w}_i^r(t)]_k$'s from $\mathcal{N}_j^*(r,k,t)$ to update its local variable. Note that there might still be $[\mathbf{w}_i^r(t)]_k$'s received from malicious nodes in $\mathcal{N}_j^*(r,k,t)$. We will see later, however, that this does not effect the workings of the overall algorithm.

The final step of the inner loop in ByRDiE is the update step (Step 8). Define $\widehat{f}(\mathbf{w}, \mathbf{Z}_j) = \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{w}, \mathbf{z}_{jn})$. Then, let $[\nabla\widehat{f}(\mathbf{w}_j^r(t), \mathbf{Z}_j)]_k$ be the k-th element of $\nabla\widehat{f}(\mathbf{w}_j^r(t), \mathbf{Z}_j)$, we can write this update step as follows:

$$[\mathbf{w}_j^r(t+1)]_k = \frac{1}{|\mathcal{N}_j| - 2b + 1} \sum_{i\in\mathcal{N}_j^*(r,k,t)\cup\{j\}} [\mathbf{w}_i^r(t)]_k - \rho(r+t-1)[\nabla\widehat{f}(\mathbf{w}_j^r(t), \mathbf{Z}_j)]_k, \tag{4}$$

where $\{\rho(\tau)\}_{\tau=1}^{\infty}$ are square-summable (but not summable), diminishing stepsizes: $0 < \rho(\tau+1) \leq \rho(\tau)$, $\sum_\tau \rho(\tau) = \infty$, and $\sum_\tau \rho^2(\tau) < \infty$. Notice that $[\mathbf{w}_j^r(T+1)]_k$ is updated after the k-th subproblem of coordinate descent in iteration r finishes and it stays fixed until the start of the k-th subproblem in the $(r+1)$-th iteration of coordinate descent. An iteration r of the coordinate descent loop is considered complete once all P subproblems within the loop are solved. The local variable at each node j at the end of this iteration is then denoted by $\mathbf{w}_j^{r,T}$ (Step 12). We also express the output of the whole algorithm as $\{\mathbf{w}_j^{\bar{r},T}\}_{j\in\mathcal{J}'}$. Note that while Algorithm 1 cycles through P coordinates of the optimization variables in each iteration r in the natural order, one can use any permutation of $\{1, \ldots, P\}$ in place of this order. The parameter T can take any value between 1 and ∞ which trades off consensus among the nonfaulty nodes and the convergence rate. We conclude by pointing out that ByRDiE has certain limitations such as stringent topology constraints, low communication efficiency and high local computing cost. As the goal of this paper is to establish the PAC learnability, we leave such improvements of the algorithm to future works.

3.2. PAC output of the algorithm

We now show that the output of the algorithm is probably approximately correct. Due to space limitations, here we only give the key steps of the analysis and intuitively explain the results. Readers may refer to an extended version [18] for more details.

The idea of coordinate descent is to sequentially minimize a function along one dimension at a time and eventually reach the minimum of that function. The main structure of ByRDiE is similar to the coordinate descent process, so we first fix an r and k and analyze the algorithm's behavior in one dimension. Here we introduce a concept that is common in distributed processing, namely, consensus. In this problem, consensus means that all nonfaulty nodes (eventually) agree on the same variable $\mathbf{w}$. When consensus is achieved, the outputs at all nonfaulty nodes are equally valid and can be considered as the output of the algorithm. We can show that consensus is guaranteed in a single dimension at the end of each inner loop. Moreover, the algorithm finds the minimum of a convex combination of local empirical risks among all nonfaulty nodes. The following lemma shows that the algorithm can guarantee consensus and optimality at the end of each inner loop.

Lemma 1. *Let Assumptions 1 and 3 hold, and let the k-th subproblem of the coordinate descent loop in iteration r of ByRDiE be initialized with some $\{\mathbf{w}_j\}_{j\in\mathcal{J}'}$. Then, as $T \to \infty$, for some $\alpha_j(r,k) \geq 0$ such that $\sum_{j\in\mathcal{J}'} \alpha_j(r,k) = 1$,*

$$\forall j \in \mathcal{J}', \left[\mathbf{w}_j^{r,T}\right]_k \to \underset{w'\in\mathbb{R}}{\arg\min} \sum_{j\in\mathcal{J}'} \alpha_j(r,k)\widehat{f}(\mathbf{w}_j|_{[\mathbf{w}_j]_k=w'}, \mathbf{Z}_j).$$

Note that the weights $\alpha_j(r,k)$ are unknown and usually different for different r and k. In particular, we are not minimizing the global empirical risk along one dimension in each inner loop. So, as much as the process is similar to coordinate descent, we are not going to reach the minimum of the empirical risk as $r \to \infty$. Instead, we are going to show that it is uniformly true for all r and k that the output of each inner loop is probably approximately correct. As a consequence, although the minimum of the empirical risk is not achievable, the final output of the algorithm is still probably approximately correct.

Since the algorithm reaches consensus at the end of each inner loop, for any r and k, all nodes can initiate each inner loop with the same $\mathbf{w}$. We denote the identical initial value at the r-th outer loop and the k-th inner loop as $\tilde{\mathbf{w}}_k^r$. Then we denote the statistical risk and convex combination of local empirical risk along one dimension, respectively, as:

$$h_k^r(w') := \mathbb{E}[f(\tilde{\mathbf{w}}_k^r|_{[\tilde{\mathbf{w}}_k^r]_k=w'}, \mathbf{z})], \quad \text{and} \tag{5}$$

$$H_k^r(w') := \sum_{j\in\mathcal{J}'} \alpha_j(r,k)\widehat{f}(\tilde{\mathbf{w}}_k^r|_{[\tilde{\mathbf{w}}_k^r]_k=w'}, \mathbf{Z}_j) \tag{6}$$

for some $\alpha_j(r,k) \geq 0$ such that $\sum_{j\in\mathcal{J}'} \alpha_j(r,k) = 1$. Now for fixed r and k, define $w^\star := \underset{w'\in\mathbb{R}}{\arg\min}\, h_k^r(w')$, and $\widehat{w} := \underset{w'\in\mathbb{R}}{\arg\min}\, H_k^r(w')$. Before going to the next step, we first make a claim that all iterates $\mathbf{w}$ stay in a bounded set for all iterations.

Specifically, $\forall r, t \quad \|\mathbf{w}_j^r(t)\|_\infty < \Gamma$. Readers may refer to [18] for proof of this claim. We then give the next lemma, which shows the PAC convergence along one dimension.

Lemma 2. *Let P and M' be fixed, $\bar{r}$ be any (arbitrarily large) positive integer, and $\{\alpha_j(r,k) \geq 0, j \in \mathcal{J}'\}_{r,k=1}^{\bar{r},P}$ be any arbitrary collection satisfying $\sum_{j\in\mathcal{J}'} \alpha_j(r,k) = 1$. Define $\bar{a} := \max_{(r,k)} \sqrt{\sum_{j\in\mathcal{J}'} \alpha_j^2(r,k)}$. Then, as long as Assumptions 1 and 2 hold, we have $\forall \epsilon > 0$, $\sup_{r,k} [h_k^r(\widehat{w}) - h_k^r(w^\star)] < \epsilon$ with probability exceeding*

$$1 - 2\exp\left(-\frac{4M'N\epsilon^2}{c_1^2 M'\bar{a}^2 + \epsilon^2} + M'\log\left(\frac{c_2}{\epsilon}\right) + P\log\left(\frac{c_3}{\epsilon}\right)\right), \tag{7}$$

where $c_1 := 8C$, $c_2 := 24CM'$, and $c_3 := 24L\Gamma P$.

Lemma 2 shows that we can approximately minimize the statistical risk with high probability along one coordinate for each subproblem in the process of coordinate descent. Since the probability bound is uniformly true for all subproblems, we can then show the final output of ByRDiE is also probably approximately correct.

Lemma 3. *Let Assumptions 1, 2, and 3 hold. Then $\forall j \in \mathcal{J}'$, $\forall \epsilon > 0$, and $T \to \infty$, we have*

$$\lim_{\bar{r}\to\infty} \left[\mathbb{E}[f(\mathbf{w}_j^{\bar{r},T}, \mathbf{z})] - \mathbb{E}[f(\mathbf{w}^*, \mathbf{z})]\right] < \epsilon \tag{8}$$

with probability exceeding

$$1 - 2\exp\left(-\frac{4M'N\epsilon^2}{{c_1'}^2 M'\bar{a}^2 + \epsilon^2} + M'\log\left(\frac{c_2'}{\epsilon}\right) + P\log\left(\frac{c_3'}{\epsilon}\right)\right),$$

where $c_1' := c_1c_4$, $c_2' := c_2c_4$, and $c_3' := c_3c_4$ for $c_4 := 2PL\Gamma$, and $(\bar{a}, c_1, c_2, c_3)$ are as defined in Lemma 2.

Theorem 1 now follows from Lemma 3 by using $\bar{\mathbf{w}}$ to denote the output of the algorithm when $T \to \infty$ and $\bar{r} \to \infty$. Note that Theorem 1 does not only indicate the PAC learnability in the presence of malicious node but it also shows the value in discussing PAC learnability in such setting. Centralized learning can be shown to have a sample complexity of $\mathcal{O}(1/\sqrt{N})$. When having M nodes and N samples at each node, distributed learning in a faultless network can have a sample complexity of $\mathcal{O}(1/\sqrt{MN})$. Theorem 1 gives a sample complexity of $\mathcal{O}(\sqrt{\bar{a}^2/N})$. Since $1/M \leq \bar{a}^2 \leq 1$, there is an improvement in sample complexity compared to local training, even though some nodes cannot be trusted.

4. NUMERICAL EXPERIMENTS

We now present the results for two sets of experiments to assess PAC learnability from distributed data in the presence of malicious nodes. We compare the performance of centralized learning, distributed learning over a faultless network, and distributed learning in the presence of malicious nodes. The goal is to show that a good hypothesis can be learned even in the presence of malicious nodes.

The first experiment involves learning a linear classifier for MNIST dataset [19]. Note that the risk function in this case fully satisfies the assumptions we made during the analysis. We randomly distribute 40K data samples evenly onto a network of 20 nodes. Each pair of nodes has a probability of 0.5 to be connected to each other. We randomly pick two nodes to be malicious nodes, which broadcast a random matrix of the same size as the classifier to all their neighbors during each iteration. We check and make sure that the network satisfies Assumption 3. We run five types of experiments: (i) collect all 40K data samples together and perform centralized learning via gradient descent; (ii) run centralized gradient descent on one of the local datasets; (iii) perform distributed learning via DGD while we replace the malicious nodes with nonfaulty nodes; (iv) perform distributed learning via DGD in the presence of malicious nodes; and (v) run ByRDiE algorithm over the network while keeping the two malicious nodes in the network. We perform 20 rounds of each experiment and evaluate the performance by counting the number of times each algorithm reaches 78% accuracy on the test set of 10K samples. The results are shown in Table 1.

Table 1. Outcomes of experiments on MNIST and Iris datasets

Algorithm	**Sample Size MNIST / Iris**	**MNIST Acc.** 78%	**Iris Acc.** 95%
Centralized GD	40K / 150	100%	99%
Local GD	2K / 15	0%	55%
Faultless DGD	40K / 150	95%	97%
DGD	40K / 150	0%	0%
ByRDiE	40K / 150	80%	96%

The goal of the second experiment is to evaluate the PAC learnability under non-convex settings. We choose a simple but non-convex neural network to be the classifier and apply it on the Iris dataset [20]. Specifically, the neural network includes a 4×3 matrix with a ReLU activation followed by a 3×3 matrix. This classifier does not satisfy the assumptions we made in the analysis. We randomly distribute the 150 data samples onto a network of 10 nodes. Each pair of nodes again has a probability of 0.5 to be directly connected. We randomly pick one node to be the malicious node and it broadcasts a random matrix of the same size as the classifier to all its neighbors during each iteration. We run the same five rounds of experiments as for the first set of experiments and the performance criterion is the number of times each algorithm reaches 95% accuracy in 100 rounds. The results are again shown in Table1.

The experimental results show that PAC learning from distributed data in the presence of malicious nodes is indeed possible. We also observe a performance gap between ByRDiE and faultless distributed training, which indicates the difference in sample complexity under faultless and malicious settings. Further, ByRDiE has better performance than training with only local dataset, which agrees with our analysis that, even in the presence of malicious nodes, learning can still benefit from having more data. The fact that DGD fails in the presence of malicious nodes again justifies the motivation of discussing PAC learnablility of distributed learning when having malicious nodes.

5. CONCLUSION

We have discussed the PAC learnability from distributed data in the presence of malicious nodes. It has been shown that, when the risk function is strictly convex with Lipschitz gradient, we can still learn a hypothesis that is probably approximately correct even when a number of nodes are trying to sabotage the algorithm. Experiments on both convex and non-convex settings were performed to show the PAC learnability on distributed datasets. We conclude by noting that PAC learnability on distributed data in the presence of malicious nodes is an open problem under non-convex and non-smooth settings. The investigation of such problems is left for future works.

6. REFERENCES

[1] M. Kearns, U. V. Vazirani, and U. Vazirani, *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[2] V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. New York, NY: Springer-Verlag, 1999.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. and Trends Mach. Learning*, vol. 3, no. 1, pp. 1–122, 2011.

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

[5] W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. Optimization*, vol. 25, no. 2, pp. 944–966, 2015.

[6] M. Balcan, A. Blum, S. Fine, and Y. Mansour, "Distributed learning, communication complexity and privacy," in *Proc. Conf. Learning Theory*, 2012, pp. 26–1.

[7] A. Blum, N. Haghtalab, A. Procaccia, and M. Qiao, "Collaborative PAC learning," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 2392–2401.

[8] L. Su and N. H. Vaidya, "Fault-tolerant multi-agent optimization: Optimal iterative distributed algorithms," in *Proc. ACM Symp. Principles of Distributed Computing*, 2016, pp. 425–434.

[9] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," in *Proc. ACM Measurement and Analysis of Computing Systems*, vol. 1, no. 2, Dec. 2017, pp. 44:1–44:25.

[10] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," in *Proc. Advances in Neural Information Processing Systems*, 2018, pp. 4618–4628.

[11] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. 35th Int. Conf. Machine Learning*, vol. 80, 2018, pp. 5650–5659.

[12] E. El-Mhamdi and R. Guerraoui, "Fast and secure distributed learning in high dimension," *arXiv preprint arXiv:1905.04374*, 2019.

[13] C. Xie, O. Koyejo, and I. Gupta, "Zeno++: Robust asynchronous SGD with arbitrary number of Byzantine workers," *arXiv preprint arXiv:1903.07020*, 2019.

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

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

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

[17] S. Sundaram and B. Gharesifard, "Distributed optimization under adversarial nodes," *IEEE Trans. Autom. Control*, vol. 64, no. 3, pp. 1063–1076, 2019.

[18] Z. Yang and W. U. Bajwa, "ByRDiE: Byzantine-resilient distributed coordinate descent for decentralized learning," *IEEE Trans. Signal Inform. Proc. over Netw.*, 2019.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] D. Dua and E. K. Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml