Defenses against Byzantine Attacks in Distributed Deep Neural Networks

Qi Xia, *Student Member, IEEE,* Zeyi Tao, and Qun Li, *Fellow, IEEE*

Abstract—Large scale deep learning is trending recently, since people find that complicated networks can sometimes reach high accuracy on image recognition, natural language processing, etc. With the increasing complexity and batch size of deep neural networks, the training for such networks is more difficult because of the limited computational power and memory. Distributed machine learning or deep learning provides an efficient solution. However, with the concern of untrusted machines or hardware failures, the distributed system may suffer Byzantine attacks. If some workers are attacked and just upload malicious gradients to the parameter server, they will lead the total training process to a wrong model or even cannot converge. To defend the Byzantine attacks, we propose two efficient algorithms: FABA, a Fast Aggregation algorithm against Byzantine Attacks, and VBOR, a Variance Based Outlier Removal algorithm. FABA conducts the distance information to remove outliers one by one. VBOR uses the variance information to remove outliers with one-pass iteration. Theoretically, we prove the convergence of our algorithms and give an insight of the correctness. In the experiment, we compare FABA and VBOR with the state-of-the-art Byzantine defense algorithms and show our superior performance.

Index Terms—Byzantine Attacks, Distributed System, Deep Learning

1 Introduction

Nowadays, it is a trending idea in the machine learning area to make the neural networks deeper and more complex for better accuracy and generality [1]. There are many complicated neural networks that are proposed recently. For instance, Christian et al. proposed GoogLeNet, which includes more than four million parameters in a 22-layer convolutional neural network (CNN). ResNet152, proposed by Kaiming et al., is a 152-layer residual neural network, has been widely used in practice [2]. In ImageNet [3], ResNet152 can perform better than the accuracy of human beings. The winner team of ImageNet competition 2016 built a 1207-layer neural network. However, due to the computational power limit, training such a complicated neural network usually takes a lot of time. Besides, people always need to tune the hyperparameter to compare for the best performance, which makes the training process more time-consuming. Although hardware development makes the training faster by implementing GPU and TPU [4] in practice, it is still relatively a long time. On the other hand, a larger batch size usually helps for a stable, fast and generalized training [5]. Because of the memory limit, we cannot train a very deep neural network with large batch sizes. To maximize the batch size, we can either use a very large memory that may cost much more, or change the training process.

There has been a lot of research to solve this problem, among which the most practical one is a technology called distributed machine learning [6], [7]. Like the classic distributed system, distributed machine learning usually includes one parameter server, which receives the gradients information from each worker, aggregate the results and synchronize the updated model and assign the datasets to every worker, and several workers, which have a copy of the model in one iteration, compute the gradients on their assigned dataset and upload the computation results to parameter server. There are two advantages of distributed machine learning. Firstly, it can reduce the computation time significantly. In the training process of neural networks, we always use stochastic gradient descent, which usually contains lots of matrix computation. With many workers, we can distribute the computations to each worker and save time. Secondly, we can use large batch size in one iteration to improve training stability and generality without memory concern. The batch size is the number of training samples to work through before the model's parameters are updated. When the batch size is large, each worker will be assigned only a small portion of training samples. This will relieve the usage of memory and help for better training results.

Similar to most distributed systems, distributed machine learning may also suffer attacks from malicious workers. For example, some workers may be compromised or have hardware failures and then just upload completely wrong gradients. Then the whole training process will converge to a malicious model. Besides, right now more and more works focus on implementing distributed machine learning on the edge computing environment [8], [9], [10]. However, this environment includes many servers from unknown sources that are not always trustful. This will also lead wrong gradients attacks. We call this kind of attack as Byzantine attacks in distributed machine learning. There are many existing works about this area. Byzantine problem was first proposed by [11] in a conventional distributed system. In 2017, Blanchard et al. first explored Byzantine problems in synchronous distributed machine learning area [12]. They talked about these failures and proposed an original method called Krum. Krum defines an algorithm based on k closest gradients to give score to uploaded gradients from each worker, and selects the gradient with the lowest score as the aggregation results. Then they also explore the Byzantine problem in asynchronous distributed system [13]. Following works are usually based on median methods. In 2018, Xie et al. proposed three similar median based methods: geometric median, marginal median, mean-around-median [14]. There are some more complicated modifications of median methods such as coordinate-wise median [15], batch normalized median [16], ByzantineSGD [17].

• *Qi Xia, Zeyi Tao and Qun Li are with Department of Computer Science, The College of William and Mary, Williamsburg, VA, 23185, USA. E-mail: {qxia,ztao,liqun}@cs.wm.edu*

However, both median based methods and Krum have a common weakness. They lose a lot of gradient information to keep the convergence and correctness of the training. For example, in Krum, it only selects one gradient out of all uploaded gradients as the aggregation result. Apparently the algorithm loses lots of information because we simply discard most of the gradients. Accordingly, it has almost no improvement compared to training on a single machine even though multiple machines are used for distributed computation. Furthermore, although Krum gives an excellent convergence proof, its assumptions are too strong to satisfy in reality. Other aggregation methods on the server side are either too complicated or too slow to resist Byzantine attack.

In this paper, we proposed two very efficient algorithms, FABA, and VBOR to resist Byzantine attack and solve the problems of slow convergence and complicated algorithms in Byzantine distributed neural networks. FABA is a method that can easily control the performance by adjusting the choice of hyper parameter, but the time complexity is $O(n)$ where n is the number of workers. VBOR uses the variance to remove the outliers of uploaded gradients who run with a complexity of $O(n)$ and thus is very efficient for a large scale distributed environment, but the performance is not as good as FABA. This is because VBOR may remove some of the honest gradients, which will affect the performance. In summary, our contributions are:

- We proposed two efficient and effective algorithms, FABA and VBOR, which defend against Byzantine attacks. Our algorithms are very easy to implement and can be modified in different Byzantine settings. More importantly, our algorithms are fast to converge even in the presence of Byzantine workers. FABA can adaptively tune the performance based on the number of Byzantine workers and VBOR is efficient in large scale distributed machine learning scenarios.
- We proved the convergence and correctness of our algorithms based on Bottou's online learning structure [18]. Mainly, we proved that the aggregation gradients by our algorithms are close to the true gradients computed by only the honest workers. We also proved that the moments of aggregation gradients are bounded by the true gradients. This ensures that the aggregation gradients are in an acceptable range to alleviate the influence of the Byzantine workers.
- We simulated the distributed environment with three types of Byzantine attacks by adding artificial noise to some of the uploaded gradients. We trained LeNet [19] on MNIST dataset and VGG-16 [20], ResNet-18, ResNet-34, ResNet-50 [2] on CIFAR-10 dataset [21] in the Byzantine distributed environment and the normal distributed environment to compare their results. Experiments showed that our algorithms could reach almost the same convergence rate as the non-Byzantine cases, with merely one or two epochs behind. Compared with the Krum and GeoMedian algorithm, our algorithms are much faster and achieve higher accuracy. Besides, we also compare FABA and Krum to show the tradeoff between the accuracy and time complexity.

2 Problem Definition and Analysis

In this section, we analyze the Byzantine problem in the distributed deep neural network.

2.1 Problem Definition

In the synchronous distributed neural network, it assumes that we have n workers, $worker_1$, $worker_2$, $\cdots, worker_n$ and one parameter server PS, which handles the uploaded gradients. Each worker keeps a replicated model. In each iteration, each worker trains on its assigned dataset and uploads the gradients $g_1, g_2, \cdots, g_n$ to the PS. The PS aggregates the gradients by average or other methods and then sends back the updated weights to all the workers as follows:

$$w_{t+1} = w_t - \gamma_t A(\overline{g_1}, \overline{g_2}, \cdots, \overline{g_n}) \tag{1}$$

Here w_t and γ_t are respectively the model weights and learning rate at time t. $A(\cdot)$ is an aggregation function that is usually an average function in classic distributed neural networks. Lastly, $\overline{g_i}$ is the uploaded gradient. The Byzantine faults may occur when some workers upload their gradients. These workers upload poisonous gradients that could be caused by malicious attacks or hardware computation errors, which means the uploaded gradient $\overline{g_i}$ may not be the same as the actual gradient g_i. We call the worker who conducts Byzantine faults as Byzantine workers. The generalized Byzantine model that is defined in [12], [14] is:

Definition 1 (Generalized Byzantine Model)**.**

$$(\overline{g_i})_j = \begin{cases} (g_i)_j & \textit{if } j\textit{-th dimension of } g_i \textit{ is correct} \\ arbitrary & \textit{otherwise} \end{cases} \tag{2}$$

As most of previous literature [12], [14], [15], we assume that there are at most $\alpha \cdot n$ Byzantine workers in this distributed system where $\alpha < 0.5$. Similar to [14], we also assume that Byzantine attackers have a full knowledge of the entire system. If not, uploaded gradients from Byzantine workers are totally different from honest workers. Any outlier removal techniques can easily filter those Byzantine workers out.

2.2 Byzantine Cases

First we discuss some cases where Byzantine faults may happen. In this way, we can better understand this problem and consider how to defend it.

Some workers are under attack. Assume the index set of the workers attacked is I, so we have:

$$\overline{g_i} = \begin{cases} g_i, i \notin I \\ arbitrary, i \in I \end{cases} \tag{3}$$

In this case, only the workers in I may upload wrong gradients, and other workers upload honestly. Intuitively, if we keep checking the uploaded gradients for sufficient time, all the Byzantine workers will be detected. Check here means in PS, we do the same computation as workers do to check if they upload the right gradients. However, in the following theorem, we show that the check algorithm cannot be determined, otherwise this scheme is not Byzantine resilient.

Theorem 1. *If the check scheme does not keep checking all the time but checks for some determined-time, this scheme is not Byzantine resilient.*

Proof. Since the Byzantine workers have the full knowledge of the entire system, they also know when the check will proceed in PS. They only need to upload actual gradients when the check proceeds. In other times, they can upload anything they want to attack this system. Because the aggregation function $A(\cdot)$ here

is average function, and without loss of generality, we assume $worker_1$ is attacked, $worker_1$ only need to upload $\overline{g_1} = n \cdot r - \overline{g_2} - \cdots - \overline{g_n}$ so that the aggregation result $A(g_1, g_2, \cdots, g_n) = r$. Then from (1), $w_{t+1} = w_t - \gamma_t r$ can be any value. □

From Theorem 1, we know that to check Byzantine workers, this scheme must have random factors so that the attackers cannot get any information before uploading the gradients. Also random checks take too much useless computation. This will definitely decrease the computation speed.

Dishonest user in Edge or IoT case. In edge or IoT cases, if we want to train a big model using each user's private data, a good way to achieve this is distributed training. However, we cannot ensure the data provided by each user is honest. Some of them may upload the gradients computed by wrong data or label.

Hardware fault causes computation fault. In most of the cases, this kind of faults usually change the $(g_i)_j$ to $(\overline{g_i})_j$ by flipping some bits in memory [22]. Actually, this kind of fault happens pretty rarely in practice (around one per several months). So we can simply ignore these kinds of faults because this does not have a huge impact. In the worst case, this makes w_t totally wrong. We can think of this w_t as a new initial random weight and start training again. On the other hand, this fault may help the training jump out of the local minima. In this way, hardware fault is not a big problem.

Network communication problem. This problem happens when the network is broken down for some reasons. Because this training process is synchronous, if the gradients from one worker cannot upload normally, all the workers need to stop to wait for it. This is easy to solve by setting an updating threshold τ. If a worker cannot update after τ, its value is ignored for this iteration.

There may be other situations where Byzantine faults happen, but the most important factors are the first two cases. In the next two sections, we will make it clear of our algorithms to resist Byzantine attacks.

3 FABA Algorithm Details

In this section, we will discuss how FABA works and the convergence proof of them.

3.1 Overview

We know that if the Byzantine gradients are very close to the average of honest gradients, the attack has almost no harm. Our proposed method is based on the observation that (i) most of the honest gradients do not differ much, and (ii) attack gradients must be far away from the true gradients in order to successfully affect the aggregation results. Note that the honest gradients are computed by the average of the mini batch dataset in each honest worker. By Central Limit Theorem, as long as the mini batch size is large enough and the dataset on each worker is randomly selected, the gradients from different workers will not differ much with high probability. We propose Algorithm 1 based on these observations.

Algorithm 1 shows that in each iteration the parameter server (PS) discards the outlier gradients from the current average. Previous methods such as Krum keep only one gradient no matter how many Byzantine workers are present, which significantly impacts the performance. Our algorithm, instead, can easily adjust the number of discarded gradients based on the number of Byzantine workers. That is, the performance will improve when the number of Byzantine workers is small.

Algorithm 1 FABA (PS Side)

Input:
The gradients computed from $worker_1$, $worker_2$, $\cdots$, $worker_n$: $G_g = \{g_1, g_2, \cdots, g_n\}$;
The weights at time t: w_t;
The learning rate at time t: γ_t;
The assumed proportion of Byzantine workers: α;
Initialize $k = 1$.

Output:
The weights at time $t+1$: w_{t+1}.

1: If $k < \alpha \cdot n$, continue, else go to Step 5;
2: Compute mean of G_g as g_0;
3: For every gradient in G_g, compute the difference between g_0 and it. Delete the one that has the largest difference from G;
4: $k = k + 1$ and go back to Step 1;
5: Compute the mean of G_g as the aggregation result at time t A_t;
6: Update $w_{t+1} = w_t - \gamma_t \cdot A_t$ and send back the updated weights w_{t+1} to each worker.

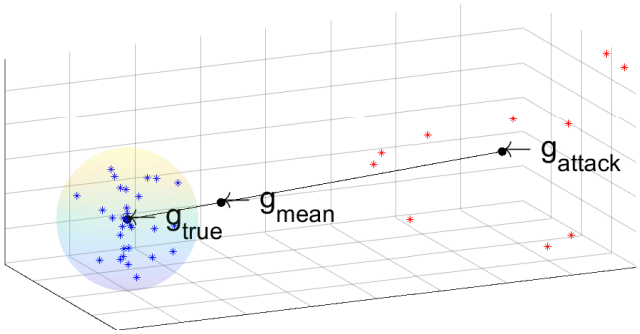

Fig. 1: Uploaded Gradients Distribution

3.2 Convergence Guarantee

Next we show that Algorithm 1 can ensure that the aggregation results are close to the true gradients. Mathematically, we have Lemma 1.

Lemma 1. *Denote honest gradients as $g_1, g_2, \cdots, g_m$ and Byzantine gradients as $a_1, a_2, \cdots, a_k$ and $m + k = n$. Let $g_{true} = \frac{1}{m}\sum_{i=1}^{m} g_i$. If we assume that $\exists \epsilon > 0, ||g_i - g_{true}|| < \epsilon$ for $i = 1, 2, \cdots, m$. Then after the process from Algorithm 1, the distance between the remaining gradients and g_{true} is at most $\frac{\epsilon}{1-2\alpha}$.*

Proof. As Figure 1 shows, the blue stars are honest gradients, and the red stars are Byzantine gradients. g_{attack} is defined as the average of the attack gradients, i.e., $g_{attack} = \frac{1}{k}\sum_{i=1}^{k} a_i$. Here, g_{mean} is the mean of all uploaded gradients from workers, i.e., $g_{mean} = \frac{1}{n}(\sum_{i=1}^{m} g_i + \sum_{i=1}^{k} a_i)$. In Figure 1, all the blue stars lie in the ball with the center of g_{true} and radius of ϵ. It is obvious that we can compute g_{mean}, but we do not know the value of g_{true} and g_{attack}.

We first compute the position of g_{mean}. It is apparent that g_{mean} lies on the line connecting g_{true} and g_{attack}. Because the assumption that the proportion of Byzantine workers is no more than α, here we assume that the number of Byzantine workers is exactly $\alpha \cdot n$, so $g_{mean} = (1-\alpha) \cdot g_{true} + \alpha \cdot g_{attack}$. Denote the distance between g_{true} and g_{attack} is l, then the distance between

g_{mean} and g_{true} is $\alpha \cdot l$ and the distance between g_{mean} and g_{attack} is $(1-\alpha)\cdot l$.

Let us talk about two cases here:

- If $l > \frac{\epsilon}{1-2\alpha}$, this is equivalent to

$$\alpha \cdot l + \epsilon < (1-\alpha)\cdot l \tag{4}$$

(4) means the $\overline{g_{mean}g_{attack}}$ is larger than $\overline{g_{mean}g_{true}}+\epsilon$. In the description of Algorithm 1, we are going to delete gradient from one worker which is farthest from g_{mean}. In this case, because all gradients in the ball are closer to g_{mean} than $\alpha\cdot l+\epsilon$, and as we know, g_{attack} is the average of all attack gradients, $\exists i \in \{1,2,\cdots,k\}$ s.t.

$$||a_i - g_{mean}|| \geq ||g_{attack} - g_{mean}|| > \alpha\cdot l + \epsilon \tag{5}$$

(5) means in this case, the gradient we delete is from Byzantine worker.

- If $l < \frac{\epsilon}{1-2\alpha}$, we cannot ensure whether gradients that we delete are from Byzantine workers. However, we can guarantee that if we delete gradients from honest workers, remaining gradients are no more than $\frac{\epsilon}{1-2\alpha}$ from g_{true}, because the $\overline{g_i g_{mean}} < \alpha\cdot l+\epsilon$, if we delete gradients from an honest worker rather than from an Byzantine worker, the distance between gradients of Byzantine worker and g_{mean} must be smaller than $\alpha\cdot l+\epsilon$. In this case, all remaining gradients are within a ball with the center as g_{true} and the radius as $l < \frac{\epsilon}{1-2\alpha}$.

Combining these two cases, we have the conclusion that gradients we delete must (i) come from Byzantine workers or (ii) come from an honest worker, but all remaining gradients are within $\frac{\epsilon}{1-2\alpha}$ distance to g_{true}.

As we repeat this process $\alpha\cdot n$ times, if gradients we delete are only from Byzantine workers, all gradients remaining are from honest workers. Otherwise, if one of gradients we delete is from Byzantine workers, then all remaining gradients are in such a ball as described before. □

Lemma 1 ensures that aggregation results from uploaded gradients are close to true gradients; $\frac{\epsilon}{1-2\alpha}$ is similar to ϵ when α is not very close to 0.5. This intuitively ensures the convergence of Algorithm 1. But to prove it, next we also need to guarantee that lower order moments of aggregation results are limited by true gradients. Theoretically, we have Lemma 2.

Lemma 2. *Let aggregation results that we get from Algorithm 1 at time t are A_t and denote G as the correct gradients estimator. If we assume $\epsilon < C\cdot||G||$ while C is a small constant, for $r = 2,3,4$, $E||A||^r$ is bounded above by a linear combination of terms $E||G||^{r_1}, E||G||^{r_2}, \cdots, E||G||^{r_l}$ with $r_1+r_2+\cdots+r_l = r$ and $l \leq n - \lceil\alpha\cdot n\rceil + 1$.*

Proof. After we proceed Algorithm 1, we delete $\alpha\cdot n$ gradients; assume that gradients left are $g^{(1)}, g^{(2)}, \cdots, g^{(p)}$ and $p = n - \lceil\alpha\cdot n\rceil$. From Lemma 1, we have $||g^{(i)} - g_{true}|| \leq \frac{\epsilon}{1-2\alpha}$ for $i \in \{1,2,\cdots,p\}$. We know from Algorithm 1 that

$$||A_t|| = ||\frac{1}{p}\sum_{i=1}^{p} g^{(i)}|| \tag{6}$$

There are at most $\alpha\cdot n$ attack gradients left here. Without loss of generality, we assume that the last $\alpha\cdot n$ gradients are from attack workers. By triangle inequality, (6) is bounded by

$$\begin{aligned}||A_t|| \leq & ||g^{(1)}|| + \cdots + ||g^{(p-\lceil\alpha\cdot n\rceil)}|| + ||g^{(p-\lceil\alpha\cdot n\rceil+1)}|| \\ & + \cdots + ||g^{(p)}|| \\ \leq & ||g^{(1)}|| + \cdots + ||g^{(p-\lceil\alpha\cdot n\rceil)}|| + \\ & ||g_{true}|| + \frac{\epsilon}{1-2\alpha} + \cdots + ||g_{true}|| + \frac{\epsilon}{1-2\alpha} \\ \leq & ||g^{(1)}|| + \cdots + ||g^{(p-\lceil\alpha\cdot n\rceil)}|| + \\ & ||g_{true}||\cdot\lceil\alpha\cdot n\rceil + C_1\cdot||G||\end{aligned}$$

So we have

$$\begin{aligned}||A_t||^r \leq C_2 \sum_{r_1+\cdots+r_{q+1}=r} & ||g^{(1)}||^{r_1}\cdots||g^{(p-\lceil\alpha\cdot n\rceil)}||^{r_{p-\lceil\alpha\cdot n\rceil}}\cdot \\ & ||g_{true}||^{r_{p-\lceil\alpha\cdot n\rceil+1}}\cdots||g_{true}||^{r_p}||G||^{r_{p+1}}\end{aligned}$$

We make an expectation on both sides, and get

$$E||A_t||^r \leq C_2 \sum_{r_1+\cdots+r_{q+1}=r} ||G||^{r_1}\cdots||G||^{r_{p+1}} \tag{7}$$

Here $r = 2,3,4$ and C_1, C_2 are two constants. □

Now we have the convergence of Algorithm 1.

Theorem 2. *We assume that (i) the cost function $cost(w)$ is three times differentiable with continuous derivatives and non-negative; (ii) the learning rates satisfy $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$; (iii) the gradients estimator satisfies $EG = \nabla Cost(w)$ and $\forall r = 2,3,4, E||G||^r \leq A_r + B_R||w||^r$ for some constants A_r, B_r; (iv) $\epsilon < C\cdot||G||$ and C is a relatively small constant that is less than 1; (v) let $\theta = \arcsin\frac{\epsilon}{(1-2\alpha)g_{true}}$, beyond the surface $||w||^2 > D$, there exists $e > 0$ and $0 \leq \psi < \frac{\pi}{2} - \theta$, s.t.*

$$\begin{aligned}||\nabla Cost(w)|| &\geq e > 0 \\ \frac{\langle w, \nabla Cost(w)\rangle}{||w||\cdot||\nabla Cost(w)||} &\geq \cos\psi\end{aligned}$$

Then the sequence of gradients $\nabla Cost(w_t)$ converges almost surely to 0.

This proof follows the proof in [18]. We use the same online learning structure to prove the convergence in non-convex settings. Basically, the idea is to first prove the global confinement of the weight, then we can use this property to prove the convergence. The detailed proof is in Appendix.

3.3 Remarks

First, in our assumption, we assume $\epsilon < C\cdot||G||$ and C is a relatively small constant. This condition guarantees that all the gradients from the honest workers gather together and their difference is small. This condition is easy to satisfy when the dataset that each worker gets is uniformly chosen and batch size is not very small. In most cases that distributed training implements, the dataset is given by the PS and each worker gets one slice of the entire dataset, thus it is almost uniformly distributed. However, in other distributed training scenarios, such as different workers keeping their own secret datasets, the distribution of the datasets is unknown. As a result of that, each dataset can be biased, and thus condition (iv) is not necessarily satisfied. We leave this for future work.

Second, we proved that the remaining gradients processed after Algorithm 1 is within $\frac{\epsilon}{1-2\alpha}$ to the true average gradient in Lemma 1, so after taking the average, the aggregation results are also within $\frac{\epsilon}{1-2\alpha}$ to it. Note that each honest worker is within ϵ distance and each honest worker can get convergence on their own. This intuitively shows the correctness of our algorithm. In fact, if the Byzantine worker ratio is less than $\frac{1}{4}$, this radius becomes 2ϵ. If the ratio is less than $\frac{1}{8}$, this radius is $\frac{4}{3}\epsilon$. This is very close to ϵ. In practice, the ratio of Byzantine workers is usually not high, which means our algorithm has good performance in these scenarios.

Third, if we combine the first two remarks and $\theta = \arcsin\frac{\epsilon}{(1-2\alpha)g_{true}}$, θ must be small here. In Figure 5, we know that condition (v) ensures that the angle between w_t and g_{true} is less than a fixed ψ that $\psi < \frac{\pi}{2} - \theta$. Since the value of α and condition (iv) guarantee that the θ is small, condition (v) is easy to satisfy. This is different from the assumption of Krum. In fact, their assumptions are difficult to satisfy because the radius of the circle is too large since it is related to the number of the dimensions in weights. In our algorithm, condition (v) becomes similar to the condition (iv) in Section 5.1 in [18], which guarantees that beyond a certain horizon, the update terms always move w_t closer to the origin on average.

In the end, our algorithm keeps $(1-\alpha)n$ gradients to aggregate. This maintains more information than previous methods. Moreover, in practice, if we do not know the number of Byzantine workers, we can simply change the number of iterations adaptively in Algorithm 1 to test whether we have the right estimate. In the beginning, we can choose a small α for better performance. When it seems to have more Byzantine workers, correspondingly we can increase α to tolerate more Byzantine attacks. This can be done during the training process, making it more flexible to balance the tradeoff between performance and correctness. Besides, we can fix the $\alpha = 0.5$. This can make the aggregation always correct.

4 VBOR Algorithm Details

In this section, we will discuss how VBOR works and the convergence proof of them.

4.1 Overview

While FABA is a fast and efficient algorithm, we need to compute the average after deleting each farthest gradient. Thus the time complexity easily reaches $O(n^2)$. Since we need to do it during every iteration in the training process, this time complexity cannot be ignored. On the other hand, in some practical implementations, such as edge computing or internet of things applications [23], there may be a lot of end devices running as workers. In this scenario, the n here can be really large, causing the time consumption of FABA a very high level. To reduce the time complexity to constant, in this subsection, we propose an alternative VBOR algorithm, which can save a lot of time and defend the Byzantine attackers. We describe Algorithm 2 below.

The idea of Algorithm 2 is to take advantage of the mean value and the standard deviation and use them to limit the range of the uploaded aggregation results. In each iteration, we limit the distance from the aggregation result and the sample average by the sample standard deviation. Although this sample includes both honest gradients and Byzantine gradients, in the following subsection, we can prove that the aggregation results from VBOR can be close to the true gradients and show the convergence of our algorithm.

Algorithm 2 VBOR (PS Side)

Input:

The gradients computed from $worker_1$, $worker_2$, $\cdots$, $worker_n$: $G_g = \{g_1, g_2, \cdots, g_n\}$;

The weights at time t: w_t;

The learning rate at time t: γ_t.

Output:

The weights at time $t+1$: w_{t+1}.

1: Compute the standard deviation σ and mean value μ for all the gradients in G_g;
2: Initialize an empty set G_{new};
3: For every gradient in G_g, if $\|g_i - \mu\| \leq \sigma$, add g_i to G_{new};
4: Compute the mean of G_{new} as the aggregation result at time t A_t;
5: Update $w_{t+1} = w_t - \gamma_t \cdot A_t$ and send back the updated weights w_{t+1} to each worker.

4.2 Convergence Guarantee

Similar to the convergence guarantee of FABA, the key of our proof is to bound the aggregation results from the expectation of the true gradient. Then we can use the online learning structure to prove the convergence. The idea of our proof also follows this idea.

First, we work on the easy problem, i.e., that there is only one Byzantine worker. This can help us better understand this algorithm and extend to multiple Byzantine workers' case. To make this convergence guarantee clearly, first we show that if the gradient that Byzantine worker uploads is bounded by the standard deviation from the average, the influence to the aggregation result is bounded and small. Theoretically, we have the following lemma:

Lemma 3. *Assume $worker_1$ is Byzantine worker and others are normal workers, i.e., $\overline{g_1} = g_1 + \delta, \overline{g_i} = g_i$ for $i = 2, \cdots, n$. Denote $\overline{g} = \frac{1}{n}\sum_{i=1}^{n}\overline{g_i}$, $g_a = \frac{1}{n}\sum_{i=1}^{n} g_i$, and $\overline{\sigma}$ is the standard deviation vector for $\overline{g_1}, \cdots, \overline{g_n}$, i.e., the vector of standard deviations for each dimension. If $\|\overline{g_1} - \overline{g}\| \leq \|\overline{\sigma}\|$ for each dimension and $\|g_1 - g_a\|$ is bounded by ϵ, then $\|\overline{g_1} - g_a\|$ is bounded.*

Proof. We have:

$$\|\overline{g_1} - \overline{g}\| \leq \overline{\sigma} \tag{8}$$

Denote $\overline{g_1} = g_1 + \delta$ and the standard deviation vector of the correct workers $g_1, g_2, \cdots, g_n$ is σ. Here we proof for one dimension case. Higher dimension cases are similar. (8) can be rewritten as:

$$\begin{aligned}
&\|g_1 + \delta - (g_a + \frac{\delta}{n})\| \\
\leq &\sqrt{\frac{(g_1 + \delta - \overline{g})^2 + (g_2 - \overline{g})^2 + \cdots + (g_n - \overline{g})^2}{n}} \\
= &\sqrt{\sigma^2 + 2(g_1 - g_a)\frac{\delta}{n} + \frac{n-1}{n^2}\delta^2}
\end{aligned}$$

Denote $\Delta = g_1 - g_a$, previous inequality is equivalent to:

$$(\Delta + \frac{n-1}{n}\delta)^2 \leq \sigma^2 + 2\frac{\delta}{n}\Delta + \frac{n-1}{n^2}\delta^2$$

So we can derive:

$$(\frac{\sqrt{(n-1)(n-2)}}{n}\delta - \sqrt{\frac{n-2}{n-1}}\Delta)^2 \leq \sigma^2 + \frac{1}{n-1}\Delta^2$$

which is equivalent to:

$$\|\frac{\sqrt{(n-1)(n-2)}}{n}\delta - \sqrt{\frac{n-2}{n-1}}\Delta\| \leq \sqrt{\sigma^2 + \frac{1}{n-1}\Delta^2}$$

This means:

$$\|\delta\| \leq \|\frac{n}{n-1}\Delta\| + \frac{n}{\sqrt{(n-1)(n-2)}}\sqrt{\sigma^2 + \frac{1}{n-1}\Delta^2}$$

In the end, let $\eta = \frac{n}{\sqrt{(n-1)(n-2)}}$, we have:

$$\begin{aligned}\|\overline{g_1} - g_a\| &\leq \|\delta\| + \|\Delta\| \\ &\leq \|\frac{2n-1}{n-1}\Delta\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta^2} \end{aligned} \tag{9}$$

Since Δ is bounded, $\|\overline{g_1} - g_a\|$ is bound. □

With the help of Lemma 3, we can extend this bound between aggregation results and expectation of the true gradients to multiple Byzantine worker cases. Theoretically, we have Lemma 4.

Lemma 4. *Assume $worker_1, \cdots, worker_k$ are Byzantine workers and others are normal workers, i.e., $\overline{g_i} = g_i + \delta_i$ for $i = 1, 2, \cdots, k$, $\overline{g_i} = g_i$ for $i = k+1, \cdots, n$. If $\|\overline{g_i} - \overline{g}\| \leq \|\overline{\sigma}\|$ for each dimension and $\|g_i - g_a\|$ is bounded by ϵ for $i = 1, 2, \cdots, k$, then $\|\overline{g_i} - g_a\|$ is bounded.*

Proof. Denote $\Delta_i = g_i - g_a$ for $i = 1, 2, \cdots, n$ and $g_a^{(i)} = (\overline{g_1} + \cdots + \overline{g_{i-1}} + g_i + \overline{g_{i+1}} + \cdots + \overline{g_k} + g_{k+1} + \cdots + g_n)/n$ for $i = 1, 2, \cdots, k$. From Lemma 3, we know that:

$$\|\overline{g_i} - g_a^{(i)}\| \leq \|\frac{2n-1}{n-1}\Delta_i\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_i^2} \tag{10}$$

If we denote $\delta_m = \max_i \|\delta_i\|$, from triangle inequality, we have:

$$\begin{aligned}\|\overline{g_i} - g_a\| \leq& \|\overline{g_i} - g_a^{(i)}\| + \|g_a^{(i)} - g_a\| \\ \leq& \|\frac{2n-1}{n-1}\Delta_i\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_i^2} \\ &+ \|\frac{\delta_1 + \cdots + \delta_{i-1} + \delta_{i+1} + \cdots + \delta_k}{n}\| \\ \leq& \|\frac{2n-1}{n-1}\Delta_i\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_i^2} + \|\frac{n-1}{n}\delta_m\|\end{aligned}$$

Without loss of generality, we assume $l = \arg\max_i \delta_i$, then we have:

$$\|\overline{g_l} - g_a\| \leq \|\frac{2n-1}{n-1}\Delta_l\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_l^2} + \|\frac{n-1}{n}\delta_l\|$$

From triangle inequality, we have:

$$\begin{aligned}\|\overline{g_l} - g_l\| \leq& \|g_l - g_a\| + \|\frac{2n-1}{n-1}\Delta_l\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_l^2} \\ &+ \|\frac{n-1}{n}\delta_l\|\end{aligned} \tag{11}$$

(11) is equivalent to:

$$\begin{aligned}\|\delta_l\| \leq& n(\|\Delta_l\| + \|\frac{2n-1}{n-1}\Delta_l\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_l^2}) \\ =& \|\frac{3n-2}{n-1}\Delta_l\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_l^2}\end{aligned} \tag{12}$$

From the definition of l, we know that $\|\delta_i\| < \|\delta_l\|$ for $i = 1, 2, \cdots, k$. Then we have:

$$\begin{aligned}\|\overline{g_i} - g_a\| \leq& \|\overline{g_i} - g_i\| + \|g_i - g_a\| = \|\delta_i\| + \|\Delta_i\| \\ \leq& \|\frac{3n-2}{n-1}\Delta_l\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\Delta_l^2} + \|\Delta_i\| \quad (13) \\ \leq& \|\frac{4n-3}{n-1}\epsilon\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\epsilon^2} \quad (14)\end{aligned}$$

Since Δ_i and Δ_l are bounded, $\|\overline{g_i} - g_a\|$ is bound. □

Because g_a is an unbiased estimate of $E\|G\|$, while G is the distribution of the correct workers' gradient, from Lemma 4, we know that as long as a worker's uploaded gradients are bounded, no matter whether it is Byzantine, the distance between these gradients to $E\|G\|$ is also bounded.

Lemma 5. *We choose average function as $A(\cdot)$ to make aggregation, and it satisfies all the conditions from Lemma 4, if there exists an constant $C > 0$, s.t. $\epsilon < C \cdot g_a$, then we have for $r = 2, 3, 4$, $E\|A\|^r$ is bounded by a linear combination of $E\|G\|^{r_1}, \cdots, E\|G\|^{r_m}$ with $r_1 + \cdots + r_m = r$.*

Proof. We have:

$$\|A\| = \|\frac{1}{n}(\overline{g_1} + \cdots + \overline{g_k} + \cdots + g_n)\|$$

By triangle inequality,

$$\begin{aligned}\|A\| &\leq \frac{1}{n}(\|g_1\| + \cdot + \|g_n\|) + \frac{k}{n}\epsilon \\ &\leq \frac{1}{n}\sum_i \|g_i\| + \frac{kC}{n}\|g_a\|\end{aligned}$$

So we have:

$$\|A\|^r \leq C_0 \sum_{r_1 + \cdots + r_{n+1} = r} \|g_1\|^{r_1} \cdots \|g_n\|^{r_n}\|g_a\|^{r_{n+1}}$$

for proper constant C_0. This implies $E\|A\|^r$ is bounded by a linear combination of $E\|g_1\|^{r_1} \cdots E\|g_n\|^{r_n} E\|g_a\|^{r_{n+1}} = E\|G\|^{r_1} \cdots E\|G\|^{r_n} E\|G\|^{r_{n+1}}$ with $r_1 + \cdots + r_{n+1} = r$. □

From Lemma 4 and Lemma 5, similar to the proof of FABA, we have the following Theorem 3 to ensure the convergence.

Theorem 3. *We assume that (i) the cost function $cost(w)$ is three times differentiable with continuous derivatives and non-negative; (ii) the learning rates satisfy $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$; (iii) the gradients estimator satisfies $EG = \nabla Cost(w)$ and $\forall r = 2, 3, 4, E||G||^r \leq A_r + B_R||w||^r$ for some constants A_r, B_r; (iv) $\epsilon < C \cdot ||G||$ and C is a relatively small constant that is less than 1; (v) let $\theta = \arcsin(\|\frac{4n-3}{n-1}\epsilon\| + \eta\sqrt{\sigma^2 + \frac{1}{n-1}\epsilon^2})/g_{true}$, beyond the surface $||w||^2 > D$, there exists $e > 0$ and $0 \leq \psi < \frac{\pi}{2} - \theta$, s.t.*

$$\begin{aligned}||\nabla Cost(w)|| &\geq e > 0 \\ \frac{\langle w, \nabla Cost(w)\rangle}{||w|| \cdot ||\nabla Cost(w)||} &\geq \cos\psi\end{aligned}$$

Then the sequence of gradients $\nabla Cost(w_t)$ converges almost surely to 0.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3035112, IEEE Transactions on Network Science and Engineering

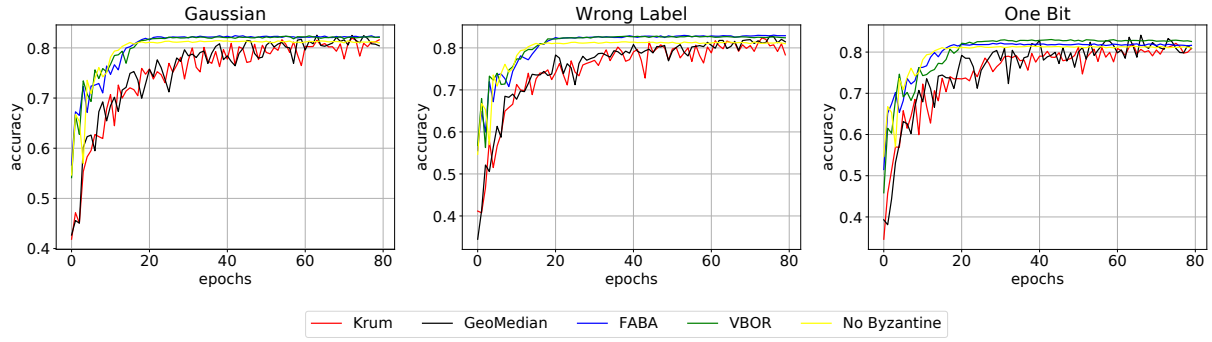

Fig. 2: Experiment results of different algorithms for Gaussian, wrong label and one bit Byzantine attacks on CIFAR-10 for 8 workers

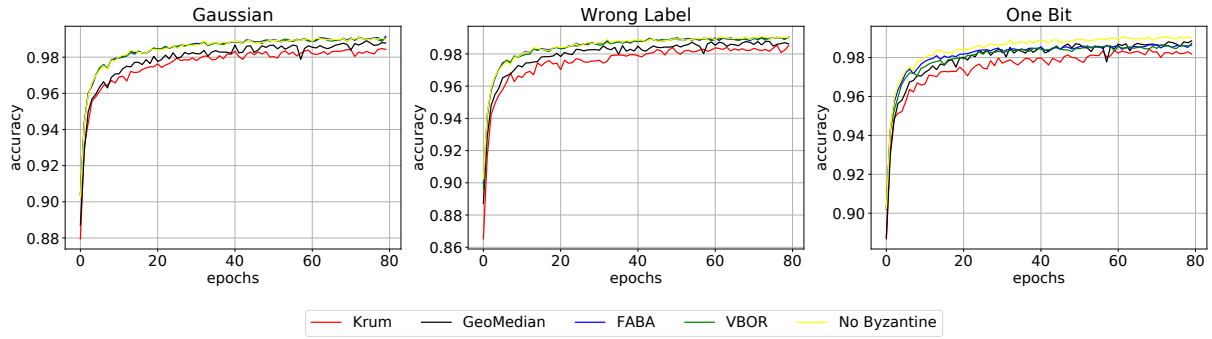

Fig. 3: Experiment results of different algorithms for Gaussian, wrong label and one bit Byzantine attacks on MNIST for 8 workers

4.3 Remarks

Similar to FABA, VBOR is also based on the gradients location information. However, there are several differences.

First, the time complexity of VBOR is $O(n)$, and the time complexity of FABA is $O(n^2)$. This does not make a very huge difference when n is small. However, in some large scale applications, such as distributed machine learning on edge computing, n can be relatively large. In this scenario, we still need to do this algorithm in PS in every iteration. The time difference can be really large.

Second, In VBOR, we also need to assume that all the true gradients are close. As we talked in Section 3.3, this needs all the assigned datasets satisfying the same distribution and the mini batch size in each worker is sufficiently large. This is very easy to achieve in practice.

Third, the convergence speed in VBOR may be not as good as FABA. In VBOR, we remove the outliers by the mean and variance value, this makes it very possible to remove byzantine gradients along with some gradients that are from the honest workers. Thus, in one iteration, FABA can keep more information than VBOR, making it converge faster than VBOR. However, FABA has one more hyperparameter than VBOR: the assumed proportion of Byzantine workers α. We need to manually tune this parameter based on the estimate of Byzantine workers during the training. Although it is possible to tune it by designing auxiliary algorithms to help tune this parameter in the training process, it is not as convenient as VBOR.

In the end, apart from using σ as the bound, in VBOR we can also change it to $c \cdot \sigma$ as the bound to remove the outliers. When the proportion of Byzantine workers is small, we can choose a large c value to accelerate the training process.

5 Experiment

In this section, we are going to run our FABA and VBOR in a simulated Byzantine environment on MNIST dataset [19] and CIFAR-10 dataset [21].

In our experiment, we conduct three different type attacks. The first is the Gaussian attack. We simply generate Gaussian noise as attack gradients and weights. The second method is wrong labeled attack. We let the label of the Byzantine workers' data randomly placed, then the Byzantine worker just normally computes the gradient and weight, and upload results with wrong labeled data to PS. The last method is one bit attack. For the uploaded gradient and weight, we only change one dimension of it with random value. For the comparison, we compare FABA and VBOR with two state-of-the-art algorithms: score-based algorithm Krum [12] and median-based algorithm GeoMedian [15]. In the experiments, we train our model on a distributed environment with 4 Nvidia GeForce GTX 1080Ti GPUs. In most of the experiments, we set the number of workers to 8, which means each GPU has 2 workers. We choose LeNet-5 [19] as the neural network to train on MNIST dataset and ResNet-18 [2] as the model to train on CIFAR-10 dataset. LeNet-5 is trained by a SGD optimizer with 0.5 as momentum and 0.01 as learning rate. ResNet-18 is trained by a SGD optimizer with 0.5 as momentum, 5×10^{-4} as weight decay and 0.01 as learning rate. The batch size for both neural networks is set to 64. We train all the experiments for 80 epochs.

5.1 Algorithm Comparison

5.1.1 8-Worker Environment

We compare FABA and VBOR with Krum, GeoMedian and no Byzantine scenario on three types of attacks that we described before for both MNIST and CIFAR-10 dataset. We have 8 total

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3035112, IEEE Transactions on Network Science and Engineering

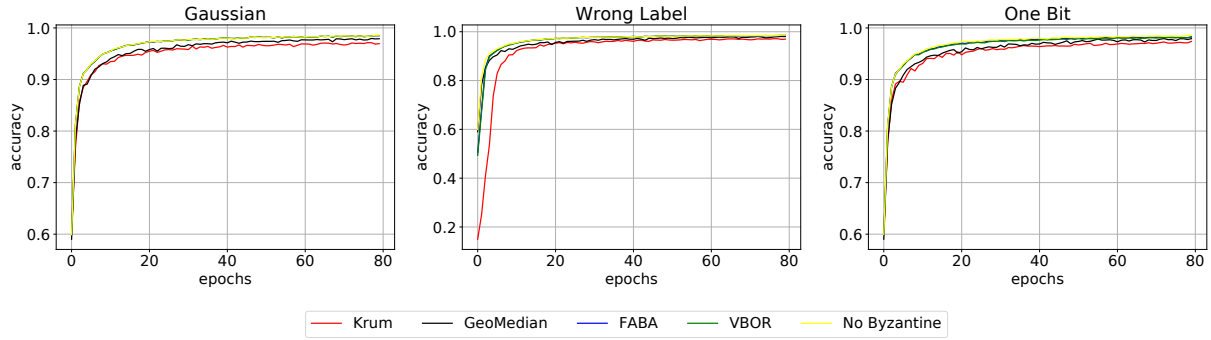

Fig. 4: Experiment results of different algorithms for Gaussian, wrong label and one bit Byzantine attacks on MNIST for 32 workers

workers, among which 2 workers are Byzantine workers. The experiment results are in Figure 2, Figure 3.

As we can see from Figure 2, FABA and VBOR converge much faster than GeoMedian and Krum, and the performance is more stable. Their performance can almost be the same or even beat a little bit compared to the no Byzantine case for all the three types of attacks. GeoMedian and Krum can also resist all the three types of attacks, but the convergence speed is much slower than our algorithms. For MNIST dataset, the results are similar. We can see from Figure 3 that FABA and VBOR outperforms GeoMedian and Krum for all kinds of attacks, while GeoMedian has slightly better performance than Krum.

5.1.2 32-Worker Environment

Because of the limitation of the hardware, we cannot deploy an environment with more workers on CIFAR-10 dataset. Thus we only deploy a 32-worker environment on MNIST dataset. Each GPU holds 8 workers. We use the same setting as the 8-worker environment. This time we changed the number of Byzantine workers to 9. The results are shown in Figure 4.

The results from Figure 4 are very similar to the 8-worker environment. FABA and VBOR have better convergence speed and performance than GeoMedian and Krum, while GeoMedian is slightly better than Krum.

5.2 Byzantine Worker Ratio Comparison

The ratio of Byzantine workers can be very different. In this section, we are going to compare the performance change of different Byzantine worker ratios. We still choose to deploy on an 8-worker environment for both CIFAR-10 and MNIST dataset. We choose the number of Byzantine workers to be 1, 2, and 3, which respectively implies the ratio of Byzantine workers 0.125, 0.25, and 0.375. The performances for the comparison between FABA, VBOR and Krum, GeoMedian are similar. So here we only show the performance of FABA and VBOR for different Byzantine worker ratios in Table 1.

	Byzantine ratio	0.125	0.25	0.375
FABA	Gaussian	99.11%	99.15%	99.09%
	Wrong Label	99.07%	99.05%	99.10%
	One Bit	98.97%	98.73%	98.25%
VBOR	Gaussian	99.11%	99.07%	99.13%
	Wrong Label	99.09%	99.04%	99.10%
	One Bit	98.87%	98.64%	98.35%

TABLE 1: The best accuracy performance of different Byzantine worker ratios on different attacks for FABA and VBOR

From this table we can see that as the Byzantine worker ratio increases, for all three types of attacks, the performance does not vary much. This shows that our algorithms are capable of defending Byzantine attacks in different ratios of Byzantine workers.

5.3 Time Complexity Comparison

We compare the time complexity for FABA, VBOR, GeoMedian and Krum on MNIST dataset and CIFAR-10 dataset. For MNIST, we use a 32 worker environment with 9 Byzantine workers. For CIFAR-10, we use an 8-worker environment with 2 Byzantine workers. The time consumption results are in Table 2.

Algorithm	Krum	GeoMedian	FABA	VBOR
MNIST				
Gaussian	11613.7s	23125.2s	4447.9s	3214.1s
Wrong Label	12025.3s	24276.2s	3697.3s	3264.5s
One Bit	11632.4s	24098.5s	4008.4s	3333.3s
CIFAR-10				
Gaussian	6291.5s	10112.7s	5245.7s	5164.8s
Wrong Label	5762.7s	9893.2s	5188.8s	5343.9s
One Bit	6663.3s	11863.9s	6080.4s	6036.0s

TABLE 2: The time complexity of different algorithms on different attacks for MNIST and CIFAR-10

From Table 2, we can see that VBOR is the most efficient algorithm compared to FABA, Krum and GeoMedian. When the number of workers is small, VBOR, FABA and Krum have similar performance on time consumption. However, when the number of workers increases, VBOR performs much better than all other algorithms, and FABA has second best efficiency performance. GeoMedian is the slowest algorithm because it adopts an iterative method to find the geometric median.

6 Conclusion

As distributed neural networks become much more popular and are being used more widely, people are beginning to enjoy the efficiency and effectiveness brought by neural networks. However, such networks are also subject to Byzantine attacks. In this paper, we proposed two effective outlier deletion based algorithms FABA and VBOR to resist the Byzantine attacks in distributed neural networks. We proved the convergence of our algorithms. In fact, in our algorithms, we can ensure that the aggregated results are very close to the true gradients. Our algorithms are more efficient because we use as much information as we can in all uploaded gradients. We take the average of all rest gradients and use it

as aggregated results. Experiments demonstrate that our algorithm can achieve approximately the similar speed and accuracy as in the non-Byzantine settings, and the performance is much better than previously proposed methods. Besides, FABA is easy to construct and control, making it simple to change how many Byzantine gradients that we want to delete by changing the parameters through the training process. VBOR has a smaller time complexity with only little accuracy loss and also is very easy to implement. We believe that our easy and original algorithms can be widely used in distributed neural networks to protect against Byzantine attacks.

ACKNOWLEDGEMENTS

This project was supported in part by US National Science Foundation grant CNS-1816399. This work was also supported in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation and workforce development. For more information about CCI, visit cyberinitiative.org.

APPENDIX
PROOF OF THEOREM 2

Proof. This proof follows Bottou's proof in [18] and the proof of Proposition 2 in the supplementary material of [12] with some modifications.

Condition (v) is complicated, so we use Figure 5 to clarify it. The dotted circle means the ball that all honest gradients lie in.

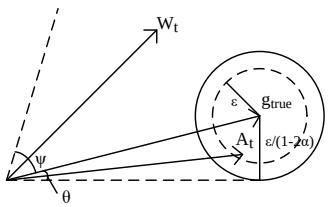

Fig. 5: Condition (v)

By Lemma 1, A_t is in the ball whose center is g_{true} and radius is $\frac{\epsilon}{1-2\alpha}$. This assumption means that the angle between w_t and g_{true} is less than ψ while $\psi < \frac{\pi}{2} - \theta$.

We start with showing the global confinement within the region $||w|| \leq D$.

(Global confinement). Let

$$\phi(x) = \begin{cases} 0 & \text{if } x < D \\ (x-D)^2 & \text{otherwise} \end{cases}$$

We denote $u_t = \phi(||w_t||^2)$.

Because ϕ has the property that

$$\phi(y) - \phi(x) \leq (y-x)\phi'(x) + (y-x)^2 \tag{15}$$

We have

$$\begin{aligned} u_{t+1} - u_t &\leq (-2\gamma_t\langle w_t, A_t\rangle + \gamma_t^2||A_t||^2)\cdot\phi'(||w_t||^2) \\ &\quad + 4\gamma_t^2\langle w_t, A_t\rangle^2 - 4\gamma_t^3\langle w_t, A_t\rangle||A_t||^2 + \gamma_t^4||A_t||^4 \\ &\leq -2\gamma_t\langle w_t, A_t\rangle\phi'(||w_t||^2) + \gamma_t^2||A_t||^2\phi'(||w_t||^2) \\ &\quad + 4\gamma_t^2||w_t||^2||A_t||^2 + 4\gamma_t^3||w_t||||A_t||^3 + \gamma_t^4||A_t||^4 \end{aligned}$$

Denote ϱ_t as the σ-algebra that represents the information in time t. We can get the conditional expectation as

$$\begin{aligned} &E(u_{t+1} - u_t|\varrho_t) \\ &\leq -2\gamma_t\langle w_t, EA_t\rangle + \gamma_t^2 E(||A_t||^2)\phi'(||w_t||^2) \\ &+4\gamma_t^2||w_t||^2E(||A_t||^2) + 4\gamma_t^3||w_t||E(||A_t||^3) + \gamma_t^4E(||A_t||^4) \end{aligned}$$

By Lemma 2, there exist positive constants X_0, Y_0, X, Y such that

$$\begin{aligned} E(u_{t+1} - u_t|\varrho_t) &\leq -2\gamma_t\langle w_t, EA_t\rangle\phi'(||w_t||^2) \\ &\quad + \gamma_t^2(X_0 + Y_0||w_t||^4) \\ &\leq -2\gamma_t\langle w_t, EA_t\rangle\phi'(||w_t||^2) \\ &\quad + \gamma_t^2(X + Y\cdot u_t) \end{aligned}$$

The first term in the right is 0 when $||w_t||^2 < D$. When $||w_t||^2 \geq D$, because of Figure 5, we have

$$\langle w_t, EA_t\rangle \geq ||w_t||\cdot||EA_t||\cdot\cos(\theta+\psi) > 0$$

So we have

$$E(u_{t+1} - u_t|\varrho_t) \leq \gamma_t^2(X + Y\cdot u_t) \tag{16}$$

For the following proof we define two auxiliary sequences $\mu_t = \prod_{i=1}^{t-1}\frac{1}{1+\gamma_i^2 Y} \xrightarrow[t\to\infty]{} \mu_\infty$ and $u'_t = \mu_t u_t$.

Because of (16), we can move $\gamma_t^2 Y\cdot u_t$ to the left and we get

$$E(u'_{t+1} - u'_t|\varrho_t) \leq \gamma_t^2\mu_t X$$

Define an indicator function χ_t as

$$\chi_t = \begin{cases} 1 & E(u'_{t+1} - u'_t|\varrho_t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Then we have

$$\begin{aligned} E(\chi_t(u'_{t+1} - u'_t)) &\leq E(\chi_t(u'_{t+1} - u'_t|\varrho_t)) \\ &\leq \gamma_t^2\mu_t X \end{aligned} \tag{17}$$

By the quasi-martingale convergence theorem [24], (17) implies that the sequence u'_t converges almost surely, which also implies that u_t converges almost surely, that is, $u_t \to u_\infty$.

If we assume $u_\infty > 0$, when t is large enough, we have $||w_t||^2 > D$ and $||w_{t+1}||^2 > D$, so (15) becomes an equality. This means that

$$\sum_{t=1}^{\infty}\gamma_t\langle w_t, EA_t\rangle\phi'(||w_t||^2) < \infty$$

Since we have $\phi'(||w_t||^2)$ converge to a positive value and in the region $||w_t||^2 > D$, by the condition (iv) and (v), we have

$$\begin{aligned} \langle w_t, EA_t\rangle &\geq \sqrt{D}||EA_t||\cos(\theta+\psi) \\ &\geq \sqrt{D}(||\nabla Cost(w_t)|| - \frac{\epsilon}{1-2\alpha})\cos(\theta+\psi) > 0 \end{aligned}$$

This contradicts the condition (ii). So we have the u_t converge to 0, which gives the global confinement that $||w_t||$ is bounded. As a result, any continuous function of w_t is bounded.

(Convergence) Here we are going to show that $\nabla Cost(w_t)$ converges almost surely to 0. First we denote $h_t = Cost(w_t)$. If we use first order Taylor expansion and bound second order derivatives with K_1, we have

$$|h_{t+1} - h_t + 2\gamma_t\langle A_t, \nabla Cost(w_t)\rangle| \leq \gamma_t^2||A_t||^2K_1 \quad a.s.$$

This implies that

$$\begin{aligned} E(h_{t+1} - h_t|\varrho_t) \leq & -2\gamma_t \langle EA_t, \nabla Cost(w_t)\rangle \\ & + \gamma_t^2 E(||A_t||^2|\varrho)K_1 \\ & \leq \gamma_t^2 K_2 K_1 \end{aligned} \tag{18}$$

This also shows that $E(\chi_t(h_{t+1}-h_t)) \leq \gamma_t^2 K_2 K_1$. By the quasi-martingale convergence theorem, h_t converges almost surely, that is, $Cost(w_t) \to Cost_\infty$. If we move the negative part to the left, take expectation of both sides and sum them for t, we get

$$\sum_{t=1}^{\infty} \gamma_t \langle EA_t, \nabla Cost(w_t)\rangle < \infty \quad a.s.$$

Next we denote $\rho_t = ||\nabla Cost(w_t)||^2$. If we use first order Taylor expansion and bound second order derivatives with K_3, we have

$$\begin{aligned} \rho_{t+1} - \rho_t \leq & -2\gamma_t \langle A_t, \nabla^2 Cost(w_t) \cdot \nabla Cost(w_t) \\ & + \gamma_t^2 ||A_t||^2 K_3 \end{aligned}$$

Taking conditional expectation of both side and bounding the second derivatives by K_4, we have

$$E(\rho_{t+1} - \rho_t|\varrho_t) \leq 2\gamma_t \langle EA_t, \nabla Cost(w_t)\rangle K_4 + \gamma_t^2 K_2 K_3$$

This implies that

$$E(\chi_t(\rho_{t+1} - \rho_t)) \leq 2\gamma_t \langle EA_t, \nabla Cost(w_t)\rangle K_4 + \gamma_t^2 K_2 K_3$$

By the quasi-martingale convergence theorem, this shows that ρ_t converges almost surely.

We have

$$\begin{aligned} & \langle EA_t, \nabla Cost(w_t)\rangle \\ \geq & (||\nabla Cost(w_t)|| - \frac{\epsilon}{1-2\alpha}) \cdot ||\nabla Cost(w_t)|| \\ \geq & (1 - \sin\theta) \cdot \rho_t \end{aligned}$$

This implies that $\sum_{t=1}^{\infty} \gamma_t \cdot \rho_t < \infty \quad a.s.$. Because condition (ii) and ρ_t converges almost surely, we have that the sequence $||\nabla Cost(w_t)||$ converges almost surely to 0.

□

REFERENCES

[1] L. J. Ba and R. Caurana, "Do deep nets really need to be deep?" *CoRR*, vol. abs/1312.6184, 2013. [Online]. Available: http://arxiv.org/abs/1312.6184

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[3] Jia Deng, Wei Dong, Richard Socher, Li-jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[4] Norman P. Jouppi, Cliff Young, and N. et al., "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 1–12.

[5] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016. [Online]. Available: http://arxiv.org/abs/1609.04836

[6] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1223–1231. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999271

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of OSDI*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899

[8] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 37–42. [Online]. Available: https://doi.org/10.1145/2757384.2757397

[9] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proceedings of the 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, ser. HOTWEB '15. USA: IEEE Computer Society, 2015, p. 73–78. [Online]. Available: https://doi.org/10.1109/HotWeb.2015.22

[10] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, "A survey of virtual machine management in edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.

[11] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[12] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 119–129. [Online]. Available: http://papers.nips.cc/paper/6617-machine-learning-with-adversaries-byzantine-tolerant-gradient-descent.pdf

[13] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous Byzantine machine learning (the case of SGD)," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1145–1154. [Online]. Available: http://proceedings.mlr.press/v80/damaskinos18a.html

[14] C. Xie, O. Koyejo, and I. Gupta, "Generalized byzantine-tolerant SGD," *CoRR*, vol. abs/1802.10116, 2018. [Online]. Available: http://arxiv.org/abs/1802.10116

[15] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," *CoRR*, vol. abs/1803.01498, 2018. [Online]. Available: http://arxiv.org/abs/1803.01498

[16] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, pp. 44:1–44:25, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3154503

[17] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," *CoRR*, vol. abs/1803.08917, 2018. [Online]. Available: http://arxiv.org/abs/1803.08917

[18] L. Bottou, "On-line learning in neural networks," in *On-line Learning and Stochastic Approximations*, D. Saad, Ed. New York, NY, USA: Cambridge University Press, 1998, pp. 9–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=304710.304720

[19] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," Department of Computer Science, University of Toronto, Tech. Rep., 2009.

[22] X. Li, M. C. Huang, K. Shen, and L. Chu, "A realistic evaluation of memory hardware errors and software system susceptibility," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855846

[23] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 68–75, 2011.

[24] M. Métivier, *Semimartingales: a course on stochastic processes*. Walter de Gruyter, 2011, vol. 2.