FABA: An Algorithm for Fast Aggregation against Byzantine Attacks in Distributed Neural Networks

Qi Xia , **Zeyi Tao** , **Zijiang Hao** and **Qun Li**
College of William and Mary, Williamsburg, VA 23185, USA
{qxia, ztao, hebo, liqun}@cs.wm.edu

Abstract

Many times, training a large scale deep learning neural network on a single machine becomes more and more difficult for a complex network model. Distributed training provides an efficient solution, but Byzantine attacks may occur on participating workers. They may be compromised or suffer from hardware failures. If they upload poisonous gradients, the training will become unstable or even converge to a saddle point. In this paper, we propose FABA, a Fast Aggregation algorithm against Byzantine Attacks, which removes the outliers in the uploaded gradients and obtains gradients that are close to the true gradients. We show the convergence of our algorithm. The experiments demonstrate that our algorithm can achieve similar performance to non-Byzantine case and higher efficiency as compared to previous algorithms.

1 Introduction

With the rapid development of deep learning, the neural network model becomes more complicated and deeper [Ba and Caurana, 2013]. For example, GoogleNet [Christian Szegedy *et al.*, 2015] contains a 22-layer deep CNN (Convolutional Neural Network) and 4 million parameters, ResNet152 [Kaiming He *et al.*, 2016] has 152 layers, which can handle more sophisticated deep learning tasks and achieve better performance than human beings on ImageNet [Jia Deng *et al.*, 2009]. Recent teams in 2016 even built a 1207-layer deep neural network on ImageNet to get better performance. However, deeper neural networks bring a huge challenge to computation due to the computation of millions of parameters. In addition, training a neural network often involves many rounds of computation based on different hyperparameters to find a better model; this process becomes even more time consuming. Although people start to use better hardware such as GPU and TPU [Norman P. Jouppi *et al.*, 2017] on tensor computation, which accelerates the training process, it is very common to take several days or weeks to train usable neural networks.

Distributed neural network is proposed to solve this problem [Dean *et al.*, 2012; Abadi *et al.*, 2016]. In neural network training, stochastic gradient descent algorithm runs on a training dataset to update a model. The batch size is the number of training samples to work through before the model's parameters are updated. People usually choose a large batch size to achieve better stability and faster convergence [Keskar *et al.*, 2016]. Indeed distributed training is well suited for training with a large batch size because it is easy to parallelize the same computation on different workers. In practice, most distributed training schemes are based on model parallelism, in which each worker keeps a copy of the model and performs the computation on their assigned dataset. A parameter server aggregates the gradients computed by the workers, updates the weights and sends back the updated weights to the workers in each iteration. When the batch size is large, e.g., 512, if we have 16 workers, each worker only needs to compute a batch size of 32. Thus, this approach will significantly reduce the computation time and the memory usage on each worker.

In a distributed neural network, one severe problem is to deal with a malicious worker, e.g., if some workers upload completely wrong gradients, what should we do to resist this kind of attack? We call networks under this kind of attacks as Byzantine distributed neural networks. Byzantine problem was first proposed by [Lamport *et al.*, 1982], and Blanchard et al. first solved this problem in a deep learning scenario [Blanchard *et al.*, 2017]. They proposed an original method called Krum, which selects a gradient from all the uploaded gradients with the lowest predefined score to resist Byzantine attacks in synchronous distributed training. They also proposed a method to resist asynchronous Byzantine attacks [Damaskinos *et al.*, 2018]. There are some other following work using various kinds of medians such as geometric median, marginal median, mean-around-median [Xie *et al.*, 2018], coordinate-wise median [Yin *et al.*, 2018; Chen *et al.*, 2017] or more complicated modification of median methods such as ByzantineSGD [Alistarh *et al.*, 2018]. However, all these methods have a common weakness: they drop a lot of useful information to keep the convergence and correctness of the training. For example, Krum selects only one gradient out of n gradients. 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.

In this paper, we proposed an efficient algorithm, FABA, to resist Byzantine attack in distributed neural networks. In summary, our contributions are:

- We proposed an efficient and effective algorithm, FABA, which defends against Byzantine attacks. Our algorithm is very easy to implement and can be modified in different Byzantine settings. More importantly, our algorithm is fast to converge even in the presence of Byzantine workers. Our algorithm can adaptively tune the performance based on the number of Byzantine workers.
- We proved the convergence and correctness of our algorithm. Mainly, we proved that the aggregation gradients by our algorithm are close to the true gradients computed by 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 Byzantine attacks by adding artificial noise to some of the uploaded gradients. We trained LeNet [Yann Lecun *et al.*, 1998] on MNIST dataset and VGG-16 [Simonyan and Zisserman, 2014], ResNet-18, ResNet-34, ResNet-50 [Kaiming He *et al.*, 2016] on CIFAR-10 dataset [Krizhevsky, 2009] in the Byzantine distributed environment and the normal distributed environment to compare their results. Experiments showed that our algorithm could reach almost the same convergence rate as the non-Byzantine cases. Compared with Krum algorithm, our algorithm is much faster and achieves higher accuracy.

2 Problem Definition and Analysis

In this section, we analyze the Byzantine problem in 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 Byzantine 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. The generalized Byzantine model that is defined in [Blanchard *et al.*, 2017; Xie *et al.*, 2018] is:

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

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

We assume that there are at most $\alpha \cdot n$ Byzantine workers in this distributed system where $\alpha < 0.5$. We also assume that the Byzantine attackers have a full knowledge of the entire system.

2.2 Byzantine Cases

We summarize the cases that Byzantine failures may occur as: (i) workers are under attacks; (ii) workers are dishonest; (iii) hardware faults cause computational faults; (iv) network communication problem. We believe that (i) and (ii) are most severe cases. Since we assume that the Byzantine workers have a full knowledge of the entire system, the Byzantine worker can manipulate the aggregated results to control the convergence. Theoretically, we have:

Theorem 1. *The byzantine worker can control the convergence of the distributed training.*

Proof. Because of (1) and that $A(\cdot)$ is an average function, without loss of generality, we assume $worker_1$ is a Byzantine worker. It only needs to upload $\overline{g_1} = n \cdot r - \overline{g_2} - \cdots - \overline{g_n}$ so that the aggregation result $A(g_1, g_2, \cdots, g_n) = r$. Therefore, it can control the training process by choosing the appropriate uploaded gradient so that the aggregated gradient becomes an arbitrary r. □

Since the Byzantine attack is harmful, we will introduce FABA to resist Byzantine attacks in the next section.

3 Algorithm Details

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

3.1 FABA 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 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.

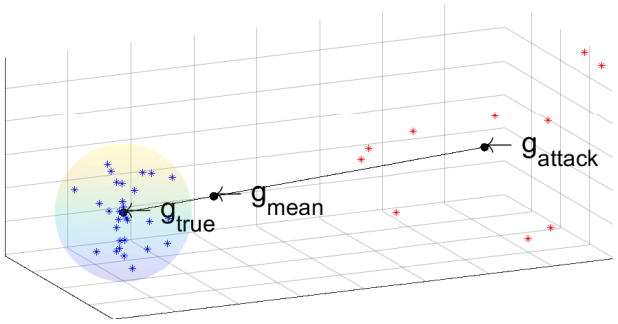


Figure 1: Uploaded Gradients Distribution

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
$$\alpha \cdot l + \epsilon < (1-\alpha) \cdot l \tag{3}$$
(3) 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 the 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.
$$||a_i - g_{mean}|| \geq ||g_{attack} - g_{mean}|| > \alpha \cdot l + \epsilon \tag{4}$$
(4) means in this case, the gradient we delete is from Byzantine worker.
- If $l < \frac{\epsilon}{1-2\alpha}$, we cannot ensure whether the gradients that we delete are from Byzantine workers. However, we can guarantee that if we delete gradients from the honest workers, the 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 the 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 the gradients we delete must (i) come from Byzantine workers or (ii) come from an honest worker, but all the remaining gradients are within $\frac{\epsilon}{1-2\alpha}$ distance to g_{true}.

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

Lemma 1 ensures that aggregation results from the 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 the lower order moments of the aggregation results are limited by true gradients. Theoretically, we have Lemma 2.

Lemma 2. *Let the 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 the 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

$$||A_t|| = ||\frac{1}{p}\sum_{i=1}^{p} g^{(i)}|| \tag{5}$$

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, (5) 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

$$E||A_t||^r \leq C_2 \sum_{r_1 + \cdots + r_{q+1} = r} ||G||^{r_1} \cdots ||G||^{r_{p+1}} \tag{6}$$

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.

Proof. This proof follows Bottou's proof in [Bottou, 1998] and the proof of Proposition 2 in the supplementary material of [Blanchard *et al.*, 2017] with some modifications.

Condition (v) is complicated, so we use Figure 2 to clarify it. The dotted circle means the ball that all honest gradients lie in. 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 $||w|| \leq D$.

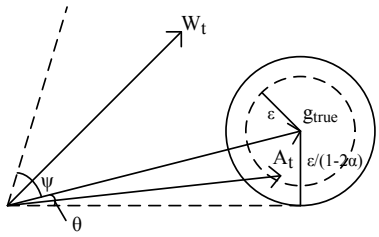


Figure 2: Condition (v)

(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

$$\phi(y) - \phi(x) \leq (y - x)\phi'(x) + (y - x)^2 \tag{7}$$

We have

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

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

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

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

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

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

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

So we have

$$E(u_{t+1} - u_t | \varrho_t) \leq \gamma_t^2 (X + Y \cdot u_t) \tag{8}$$

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 (8), 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

$$E(\chi_t(u'_{t+1} - u'_t)) \leq E(\chi_t(u'_{t+1} - u'_t|\varrho_t)) \leq \gamma_t^2 \mu_t X \quad (9)$$

By the quasi-martingale convergence theorem [Métivier, 2011], (9) 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 (7) 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) Once we proved that w_t is bounded, the rest convergence part proof is the same as Bottou's proof in [Bottou, 1998]. Thus we proved Theorem 2.

□

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 get 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 keep 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 2, 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 [Bottou, 1998], which guarantees that beyond a certain horizon, the update terms always moves 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 number of iterations k for better performance. When it seems to have more Byzantine workers, correspondingly we can increase k 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.

4 Experiment

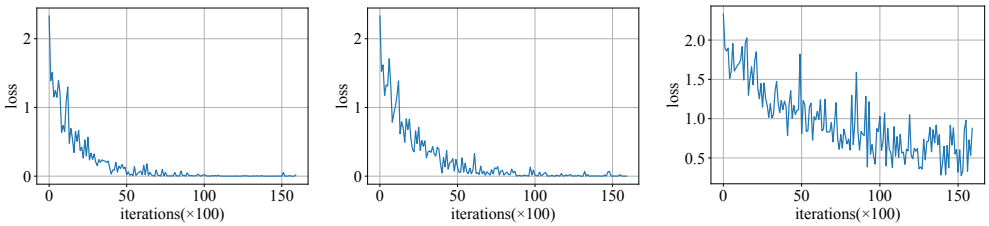
In this section, we are going to run our FABA in a simulated Byzantine environment on MNIST dataset and CIFAR-10 dataset. We chose the Byzantine proportion as 0.3, which means 30% of the total workers are under attack and upload noise as gradients. In the experiment, wo chose uniform and Gaussian distribution to generate the attack gradients. Because the results are similar, here we only show the results with uniform distribution in $(-0.25, 0.25)$. This means each coordinate in the Byzantine gradients is generated by a uniform random number from $(-0.25, 0.25)$. Here, we only compared FABA with Krum because in [Xie *et al.*, 2018], it is shown that the median methods and Krum have similar convergence rate.

4.1 MNIST

We deployed a 32-worker environment on a server with 44 cores Intel E5-4669v4 and 792 GB RAM. Because we only need to simulate the distributed environment with Byzantine workers, we did not deploy this distributed environment on several machines. We used a single machine with multiple cores to simulate this environment on CPU. Since the Byzantine proportion is 0.3, 9 out of 32 workers are Byzantine workers.

We respectively implemented our algorithm and Krum on a LeNet network in this Byzantine environment. Each worker was trained using a batch size of 4, so the total batch size is 128. Because MNIST converges quickly, we only trained 10 epochs here. For our algorithm and Krum algorithm, the results are shown in Table 1.

As shown, our algorithm converges much faster than Krum. We also plotted the loss change every 100 iterations in Figure 3. The loss change is much larger in Krum than our algorithm, which shows our algorithm is more stable than Krum. Although it finally converges, it is more unstable. This

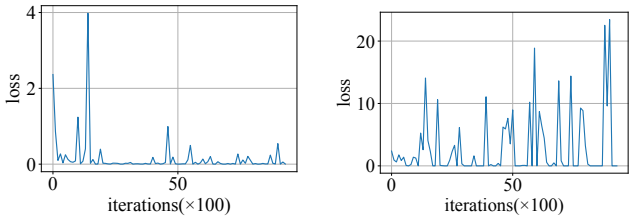


(a) No Byzantine (b) FABA (c) Krum

Figure 5: Loss change every 100 iterations

Epoch	1	2	3	4	5
FABA	0.9536	0.9736	0.9725	0.9702	0.9649
Krum	0.6814	0.7306	0.7633	0.7684	0.7811
Epoch	6	7	8	9	10
FABA	0.9632	0.9551	0.9536	0.9552	0.9529
Krum	0.7553	0.7896	0.8035	0.8063	0.8112

Table 1: Accuracy of 10 Epoches for FABA and Krum on 32-worker 0.3 proportion Byzantine rate and 0.2 noise distribution



(a) FABA (b) Krum

Figure 3: Loss change every 100 iterations

is easy to understand because it only uses the gradient from a single machine, whose batch size is 4 while our algorithm use a batch size of $4 \times (32 - 9) = 92$. A large batch size will lead to a faster and stable convergence.

4.2 CIFAR-10

We trained CIFAR-10 respectively on VGG-16, ResNet-18, ResNet-34 and ResNet-50 networks. Since these neural networks are complicated and really slow to train in CPU environment, we redeploy an 8-worker environment on a Google Cloud instance with 8 vCPUs, 30 GB memory and 8 NVIDIA Tesla K80 GPUs. Here, we still chose the Byzantine proportion as 0.3. Each worker was trained using a batch size of 32, so the total batch size was 256. Each GPU kept a same model and was thought as a worker. We used learning rate 0.01, momentum alpha 0.9 and weight decay as 5×10^{-4} for 80 epochs. We trained these models on Byzantine environment respectively using our algorithm and Krum. We also compared these results with the same distributed environment in the non-Byzantine setting. Because the results for different networks are similar, we only show the result in ResNet-18. The results are shown in Figure 4. Here we use top-1 accuracy to compare their performance.

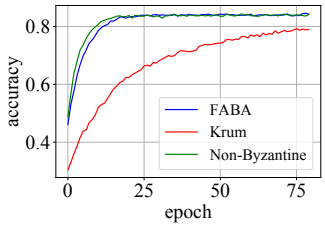


Figure 4: Accuracy

Figure 4 shows the accuracy change through the epochs. It is obvious that in this setting, our algorithm is much faster than Krum and even reaches a higher top-1 accuracy. It also shows that our algorithm achieves almost the same speed as the non-Byzantine distributed training without Byzantine attack, while there is only one or two epochs behind and the final accuracy is almost the same. Figure 5 is the loss change every 100 iterations. As we can see, our algorithm is almost the same as the case without Byzantine, while Krum is much more unstable than our algorithm. This shows that our algorithm is enough to resist Byzantine worker.

5 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 FABA to defend against Byzantine attacks in distributed neural networks. We proved the convergence of our algorithm. Experiments demonstrate that our algorithm can achieve approximately the same speed and accuracy as in the non-Byzantine settings, and the performance is much better than previously proposed methods. Additionally, FABA can extract more performance gain (by retaining more gradients) depending on the number of Byzantine workers compared with the prior work. It is easy to tune the algorithm based on the number of Byzantine workers. We believe this elegant algorithm can be widely used in distributed neural networks to defend against Byzantine attacks.

Acknowledgements

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grant CNS-1816399.

References

[Abadi *et al.*, 2016] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of OSDI*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.

[Alistarh *et al.*, 2018] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. *CoRR*, abs/1803.08917, 2018.

[Ba and Caurana, 2013] Lei Jimmy Ba and Rich Caurana. Do deep nets really need to be deep? *CoRR*, abs/1312.6184, 2013.

[Blanchard *et al.*, 2017] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 119–129. Curran Associates, Inc., 2017.

[Bottou, 1998] Léon Bottou. On-line learning in neural networks. In David Saad, editor, *On-line Learning and Stochastic Approximations*, pages 9–42. Cambridge University Press, New York, NY, USA, 1998.

[Chen *et al.*, 2017] Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(2):44:1–44:25, December 2017.

[Christian Szegedy *et al.*, 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[Damaskinos *et al.*, 2018] Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Rhicheek Patra, and Mahsa Taziki. Asynchronous Byzantine machine learning (the case of SGD). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1145–1154, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[Dean *et al.*, 2012] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1223–1231, USA, 2012. Curran Associates Inc.

[Jia Deng *et al.*, 2009] 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*, pages 248–255, June 2009.

[Kaiming He *et al.*, 2016] 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)*, pages 770–778, June 2016.

[Keskar *et al.*, 2016] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[Lamport *et al.*, 1982] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[Métivier, 2011] Michel Métivier. *Semimartingales: a course on stochastic processes*, volume 2. Walter de Gruyter, 2011.

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

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Xie *et al.*, 2018] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant SGD. *CoRR*, abs/1802.10116, 2018.

[Yann Lecun *et al.*, 1998] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[Yin *et al.*, 2018] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. *CoRR*, abs/1803.01498, 2018.