ToFi: An Algorithm to Defend Against Byzantine Attacks in Federated Learning

Qi Xia(✉), Zeyi Tao, and Qun Li

Department of Computer Science, William and Mary, Williamsburg, VA 23185, USA
{qxia,ztao,liqun}@cs.wm.edu

Abstract. In distributed gradient descent based machine learning model training, workers periodically upload locally computed gradients or weights to the parameter server (PS). Byzantine attacks take place when some workers upload wrong gradients or weights, i.e., the information received by the PS is not always the true values computed by workers. Approaches such as score-based, median-based, and distance-based defense algorithms were proposed previously, but all of them made the asumptions: (1) the dataset on each worker is independent and identically distributed (i.i.d.), and (2) the majority of all participating workers are honest. These assumptions are not realistic in federated learning where each worker may keep its non-i.i.d. private dataset and malicious workers may take over the majority in some iterations. In this paper, we propose a novel reference dataset based algorithm along with a practical Two-Filter algorithm (ToFi) to defend against Byzantine attacks in federated learning. Our experiments highlight the effectiveness of our algorithm compared with previous algorithms in different settings.

Keywords: Byzantine attacks · Federated learning

1 Introduction

Federated learning [6,11] is a collaborative machine learning scheme in which the participating machines hold their local data without exchanging them. Combined with edge computing, it has the advantages of easy implementation [14], better privacy [18,19], and communication efficiency improvement [9,10,13]. The machines (or workers and nodes) performing the training task in federated learning upload intermediate computation results to the parameter server (PS) for the aggregation and model update. In addition, federated learning sometimes only randomly selects some of the nodes to perform the computation in one synchronization round, and meanwhile, each node usually performs self-update for several intervals for this synchronization round. This significantly reduces the communication cost between nodes and the PS. Therefore, federated learning attracts more interests in both academic research and industry application.

Because federated learning is a special kind of distributed machine learning, it is naturally subject to security attacks when multiple nodes communicate

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 398, pp. 229–248, 2021.
https://doi.org/10.1007/978-3-030-90019-9_12

with each other. For example, in classic distributed machine learning, Byzantine problems exist when some nodes undergo attacks and do not perform honestly when uploading the computation results to the PS. Thus the training process will be dominated by dishonest nodes. This problem becomes even more severe in federated learning. In federated learning settings, nodes are from different resources, among which some are trusted and some can be untrusted. This is different from the classic distributed machine learning, where we assume all the nodes are in the laboratory environment and are under control. In this scenario, nodes in the federated learning system are more likely to be attacked or compromised intentionally or unintentionally. For example, many users collaborate to train an image recognition model, during which some users may upload a cat picture and mark it as a dog. This operation apparently affects the performance of the model training. When the number of such users or activities is large, the training process will lead to a wrong model.

Although there are some algorithms that work well toward solving Byzantine problems in traditional distributed machine learning, these algorithms do not perform well for federated learning due to the following two major problems:

- The distribution of the dataset on each node may be different. Data heterogeneity is an important feature of federated learning. It by design comes along with the local data isolation in each node. Because each node can keep its own private dataset, the data distribution between different nodes may be significantly different. Therefore the computational results of each node based on the local dataset can be very different such that it is difficult and sometimes impossible to distinguish between honest nodes and Byzantine nodes.
- Honest majority is not a reasonable assumption in federated learning. A typical assumption made in Byzantine machine learning problems for traditional distributed machine learning is that honest nodes are the majority, which means the number of honest nodes is more than half of the total nodes. Although we can assume the honest nodes are the majority in federated learning, in each synchronization round, we cannot assume that there are more honest nodes in the random selection. For example, we have 100 nodes in total, among which 20 nodes are malicious. In each synchronization, we randomly choose 10 nodes to do the computation. It is possible that more than 5 malicious nodes are selected. This will make all the previous algorithms ineffective because malicious nodes can take over the training. Moreover, it is hard for some of the previous algorithms to be implemented in federated learning. For example, Zeno [22] and FABA [16] need to estimate the ratio of Byzantine nodes, which is hard because the ratio changes in different synchronization rounds in federated learning.

In this paper, we carefully investigate these two challenges of Byzantine problems in federated learning and propose a naive algorithm and modify it to an efficient algorithm ToFi to defend against Byzantine attacks. In summary, our contributions are:

- We compare the major differences of Byzantine problems between federated learning and distributed learning and analyze why previous Byzantine-resilient algorithms do not work well in federated learning.
- We propose a naive algorithm to solve Byzantine problems in federated learning. More importantly, we modify it to an efficient two-filter reference dataset-based algorithm ToFi to efficiently defend against Byzantine attacks in practical implementations.
- We conduct several simulated experiments in various heterogeneous environments to compare ToFi and other existing algorithms to show our superior performance of defending against Byzantine attacks in federated learning.

2 Related Works

In order to resist Byzantine attacks in classic distributed machine learning, some algorithms have been proposed. Basically, there are three directions for defending against Byzantine attacks: score-based, median-based, and distance-based algorithms.

The idea of score-based algorithms is that there is a scoring system on the server side such that this system assigns corresponding scores for each uploaded gradient. This is the earliest idea to defend against Byzantine attacks in this area. Blanchard et al. first proposed an algorithm called Krum [2]. In Krum's design, each gradient's score is based on the summation of the distances to its nearest neighbors. Then the server simply chooses the gradient with the smallest score as the aggregation result. This gradient has the property that it is the closest one that nears its neighbors, so it should come from an honest node with high probability. However, because only one gradient is selected as the aggregation result, a lot of useful information from other uploaded gradients is missing. Therefore, the convergence speed of Krum is slow. After this, they also proposed another algorithm to resist asynchronous Byzantine attacks [4]. In addition, Xie et al. also proposed methods based on a reference dataset to give scores for each node to solve fault-tolerance problems in distributed machine learning [21,22]. Their scoring systems use the reference dataset to examine the loss of each uploaded gradient and the server finally chooses the gradients that result in smaller loss.

Later the idea of defending against Byzantine attacks has been moved to the geometric median-based algorithms. By definition, the geometric median of a discrete set of points in Euclidean space is a point that minimizes the sum of distances to the sample points. For example, Xie et al. proposed geometric median, marginal median, and median-around-median [20], Yin et al. proposed coordinate-wised median [24], Lili et al. proposed a batch normalized median [3], Alistarh et al. proposed a more complicated modification of median-based methods called ByzantineSGD [1]. The geometric median is an important estimator of location in statistics and it can preserve the majority location information of the sample points, which in Byzantine problems, represent more about the honest update information. Although median-based algorithms usually have better

convergence performance than score-based algorithms, they also have one shortcoming. The geometric median of several sample points usually needs an iterative method to solve and thus the computational time is considerable. On the server side, it takes a too long time for the aggregation, so the training speed will be slower.

Distance-bases algorithms conduct the distance information to remove the dishonest gradients. From the i.i.d. dataset and central limit theorem, the honest gradients should be close to each other. Therefore, in order to defend against Byzantine problems in this scenario, the problem is transformed to outlier gradient removal based on distance information. Xia et al. proposed an alternative method called FABA [16]. Instead of using median-based methods, they used Euclidean distance to remove outlier gradients. They adaptively remove outliers based on the distance between the current average gradient and the current remaining gradients. They later provided another Byzantine-resilient algorithm for large-scale distributed machine learning [17]. By using simple statistics of multi-dimensional mean and standard deviation, it can remove the outliers in $O(n)$ time. It has comparable performance with other algorithms while keeping a fast training speed.

3 Preliminary

3.1 Federated Learning

Here we give a brief introduction about federated learning, which is a special kind of the distributed learning. Figure 1 is the structure of one synchronization in federated learning at time t. It has one central server PS and n workers $worker_1, \cdots, worker_n$. In each iteration, the PS randomly selects m workers. The selected workers then fetch the global model from the PS and perform the computation on their local dataset. Without loss of generality, we assume $worker_1$ to $worker_m$ are selected. On the worker side, those participated workers then update their local model on their private dataset ξ_i. If we assume the loss function on the neural network is $f(\cdot)$, weight at time t is w_t and learning rate at time t is γ_t, stochastic gradient descent will update the local weight for $worker_i$ at time $t+1$ as:

$$w_{t+1}^i = w_t - \gamma_t \cdot \frac{\partial f(w|\xi_i)}{\partial w}\bigg|_{w_t} \quad (i = 1, 2, \cdots, m) \tag{1}$$

On the PS side, it receives the updated weights uploaded by all the participated workers. PS uses an aggregation function $A(\cdot)$ to aggregate the uploaded weights and update the global model.

$$w_{t+1} = A(w_{t+1}^1, w_{t+1}^2, \cdots, w_{t+1}^m) \tag{2}$$

In practice, we usually simply use a weighted average function to aggregate the uploaded weights.

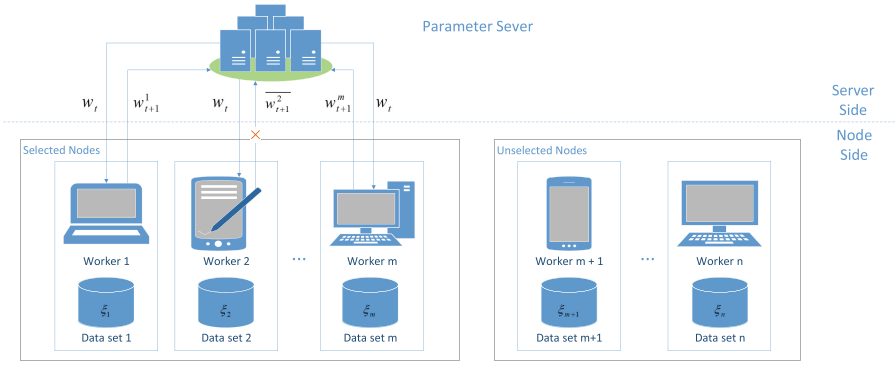

Fig. 1. Federated learning with Byzantine attackers at time t

3.2 Byzantine Problem

Byzantine problems exist when some workers are attacked or compromised and do not compute or upload weights correctly. In this scenario, the uploaded weights w_{t+1}^i in (2) may not be the real w_{t+1}^i computed by (1). Theoretically, the generalized Byzantine model that is defined in [2,20] is:

Definition 1 (Generalized Byzantine Model)

$$\overline{w_{t+1}^i} = \begin{cases} w_{t+1}^i & if\ i\text{-}th\ worker\ is\ honest \\ a_i \neq w_{t+1}^i & otherwise \end{cases} \tag{3}$$

Here we denote $\overline{w_{t+1}^i}$ as the actual weights received by PS from $worker_i$. In each iteration of the training phase, some workers may become Byzantine workers and upload attack weights a_i to the PS. As we can see in Fig. 1, $worker_2$ here is a Byzantine attacker. It uploads an alternative $\overline{w_{t+1}^2}$ rather than the actual w_{t+1}^2 to the PS. The PS, at the same time, does not know $worker_2$ is compromised. It aggregates all the uploaded weights and sends the incorrectly updated weights back to all workers. According to Theorem 1 in [16], when the aggregation function is an average function, one Byzantine attacker can take over the aggregation result and lead the whole training process to an incorrect phase.

3.3 Discussions About Byzantine Problems in Federated Learning

In the introduction, we have discussed two major differences between traditional distributed machine learning and federated learning. These two differences make it extremely difficult to solve Byzantine problems in federated learning. For example, if we assume the number of Byzantine nodes is equal to the number of honest nodes and let Byzantine nodes have the same data as honest nodes with different labels, the PS is not able to distinguish which training results are real.

Thus it is not able to solve Byzantine problems without other information. In addition, when the dataset is not i.i.d. and the computational results are totally different from each other, it is very hard for the PS to distinguish between honest results and Byzantine results.

Therefore, in order to solve Byzantine problems in federated learning, more conditions are needed. Here like [7] and [21,22], we assume that PS holds a small reference dataset to solve Byzantine problems in federated learning. Our theoretical assumptions are defined in Sect. 4.

4 Problem Definition

Let D and D_i respectively represent the distribution for the whole dataset and each worker. We first assume that the distributed environment is heterogeneous.

Assumption 1 (Heterogeneous environment). *Updated weights of each worker are computed based on their private non i.i.d. dataset, i.e.,*

$$w_{t+1}^i = w_t - \gamma_t \cdot \frac{\partial f(w|\xi_i)}{\partial w}\bigg|_{w_t}, \quad \xi_i \sim D_i \quad (i = 1, 2, \cdots, m) \tag{4}$$

Here m is the number of selected workers in the federated learning system.

The second assumption is for the Byzantine environment. The Byzantine model is given by Definition 1.

Assumption 2 (Byzantine environment). *Denote $B_t = \{i|\overline{w_{t+1}^i} \neq w_{t+1}^i\}$. We have: (i) B_t can be different from each other; (ii) $0 \leq |B_t| \leq n$.*

Assumption 2 states two features of Byzantine attacks. First, Byzantine attacks can target any machine in the distributed system and can change target machines during the training process. This means that different sets of workers may be attacked during different iterations of the training. Second, there is no upper limit of the number that Byzantine attacks may happen in one iteration, i.e., in one iteration, it is possible that the majority of workers suffer Byzantine attacks.

Assumption 3 (Reference dataset). *PS holds a small reference dataset ξ_R such that $\xi_R \sim D$.*

Assumption 3 is reasonable because, in practice, PS is usually able to collect some data. The size of the reference dataset can be very small, for example, 100 to 500 is good enough, so we can assume PS has a reference dataset. We will talk about the existence of the reference dataset in Sect. 5.1.

With these three assumptions, the problem is finding an appropriate aggregation algorithm $A^*(\cdot)$ and using (2) to update weights until convergence point w_c such that

$$A^* = \arg\min_{A(\cdot)} f(w_c)$$

It is obvious that a simple average is not able to resist Byzantine attacks. Our goal is to find a Byzantine robust aggregation algorithm in federated learning.

5 Algorithm

5.1 Naive Algorithm

Because the PS has no other information about the dataset but only uploaded weights, it must have some extra information to filter and aggregate weights. The key of our algorithm is that PS holds a small reference dataset. As this reference dataset has the assumption that $\xi_R \sim D$, this guarantees that the reference dataset has the same distribution as the whole dataset and the most information about it. The whole dataset here is a pretty tricky definition. Since some workers are compromised, their subdatasets are inaccessible by the PS. This implies that the accessible dataset may only contain part of the whole dataset. However, we still take the whole dataset as the optimization goal, and the reference dataset is chosen from this distribution even though some data are hidden by Byzantine attackers. Because the PS collects reference dataset on its own and PS and all nodes share the same training goal, it is reasonable to assume there exists such reference dataset. We will use the reference dataset to examine the weights $\overline{w_{t+1}^i}$ uploaded by each worker and aggregate them to update the weights for each iteration.

The naive algorithm is described in Algorithm 1. Because we do not have access to each worker, our algorithm is only running on the PS side. In the PS, its inputs are the weights $\overline{w_{t+1}^1}, \overline{w_{t+1}^2}, \cdots, \overline{w_{t+1}^m}$ that are computed and uploaded by the selected workers. The output in one iteration should return the updated weights to all the workers.

Algorithm 1 Reference dataset based naive algorithm (PS Side)

Input:

Weights computed from $worker_1$, $worker_2$, $\cdots$, $worker_m$: $G_w = \{\overline{w_{t+1}^1}, \overline{w_{t+1}^2}, \cdots, \overline{w_{t+1}^m}\}$;
The reference dataset ξ_R.

Output:

Weights at time $t+1$: w_{t+1}.

1: Solve α_i by minimizing α-weighted loss: $\alpha_i = \arg\min_{\sum \alpha_i = 1} f(\sum_{i=1}^m \alpha_i \overline{w_{t+1}^i}, \xi_R)$;
2: Aggregate weights by $w_{t+1} = \sum_i^m \alpha_i \overline{w_{t+1}^i}$;
3: Send back w_{t+1} to each worker.

Assume the weights at $t+1$ is w_{t+1}, it should be updated by w_t using the α-based weighted average. In each step, we minimize the loss on the reference dataset ξ_R to get the corresponding α_i. We have:

$$\arg\min f(w_{t+1}, \xi_R) = \arg\min_{\alpha_i} f(\sum_{i=1}^{m} \alpha_i \overline{w_{t+1}^i}, \xi_R) \tag{5}$$

It should be noted that since $\xi_R \sim D$ while D stands for the distribution of the whole dataset, we can think that $f(\cdot, \xi_R)$ and $f(\cdot, \xi)$ are similar. Therefore our goal is going to minimize (5) to compute α_i.

Let us take a look at the (5). In fact, $\sum_{i=1}^{m} \alpha_i \overline{w_{t+1}^i}$ is a linear combination of $\overline{w_{t+1}^i}$. If we denote each $\overline{w_{t+1}^i}$ as a coordinate in a $(m-1)$-dimensional space $\mathbb{W}$, then solving α_i transforms to finding a point in the space $\mathbb{W}$ that minimizes a function $f(\cdot|w_t^i, \xi_R)$. We can use several existing techniques to find this minimal point. Grid method is an intuitive way with slow but effective performance. It divides the space into several grid points and tries to find a point with the lowest loss function value. However, its time complexity increases exponentially with the dimension. We can also use the classic gradient descent to find the optimization solution.

In addition, we can show the intuition of the correctness of this algorithm. In each iteration, we are going to choose the best updated direction of the weights on the reference dataset. Because this direction is an α-weighted average of all uploaded weights and it minimizes the loss function, assuming we have solved α_i, we have:

$$f(\sum_{i=1}^{m} \alpha_i \overline{w_{t+1}^i}, \xi_R) \leq f(\overline{w_{t+1}^k}, \xi_R) \tag{6}$$

This is because α_i is obtained by minimizing the loss function. We can get (6) by setting the α-weight of $w_t^{(k)}$ to be 1 and all the other α-weights to be 0. (6) shows that at least the performance of each iteration on the reference dataset is better than any worker, including honest workers and Byzantine workers. Mathematically, using the same idea, we can prove the following lemma:

Lemma 1. *The α-weights that we get from Algorithm 1 Step 1 introduce the smaller loss on the reference dataset than just ideally taking the average of all the honest weights in each iteration.*

Proof. Without loss of generality, denote the first p workers are honest and the rest are attack workers, i.e., $w_{t+1}^1, w_{t+1}^2, \cdots, w_{t+1}^p$ are true weights from honest workers and $w_{t+1}^{p+1}, w_{t+1}^{p+2}, \cdots, w_{t+1}^m$ are true gradients from dishonest workers. Then the loss of only taking the average of honest weights $loss_{honest}$ is:

$$loss_{honest} = f(\sum_{i=1}^{p} \frac{1}{p} w_{t+1}^i, \xi_R) \tag{7}$$

By the definition of α-weights, we have:

$$f(\sum_{i=1}^{m} \alpha_i \overline{w_{t+1}^i}, \xi_R) \leq loss_{honest} \tag{8}$$

This is because the right of (8) is obtained by letting $\alpha_i = \frac{1}{p}$ for $i = 1, 2, \cdots, p$ and all the other $\alpha_i = 0$. Since $\xi_R \sim D$, α-weighted average has smaller loss in each iteration.

We know that even if we only take the weights from one of the honest workers, the training can still converge to a reasonable model. When we have more workers, although there are some Byzantine workers, the loss function examination on the reference dataset and α-weighted average can help to get a reasonable model.

5.2 ToFi Algorithm

With the help of the reference dataset, the naive algorithm can defend against Byzantine attacks. However, this naive solution is hard to implement in practice. First of all, this method relies too much on the reference dataset. Therefore, it is more like searching minimal points on the space of the reference dataset using the computation weight projections from the local dataset to this space. Secondly, this naive method is lacking in solving the case that in some iterations, all of the participated nodes are attacked and become malicious. Thirdly, it is time-consuming to solve the optimization problem in each synchronization.

In order to mitigate these problems, we modify the naive algorithm to our reference dataset-based two-filter algorithm ToFi. The core ideas of ToFi are below. First, in order to approximate the α-weight in the naive algorithm, we adopt the softmax function of the examined loss on the reference dataset for each worker so that the worker with a smaller loss will get a larger α-weight. Second, in order to deal with the abnormal loss received by the PS, we adopt two filters based on the normalized loss and update similarity to remove outlier updates. The details of our enhanced algorithm are described in Algorithm 2.

Let us take a look at Algorithm 2. It is an adaptive way to compute the aggregated weight in a naive algorithm. In summary, there are two filters: reference dataset-based loss filter and update similarity-based filter. Those filters are designed to remove outlier weights. The detailed discussions are below.

Reference dataset-based loss filter. The reference dataset here is used to examine the performance of the uploaded weights computed by each node from their private dataset. Because of the heterogeneity of the data distribution, the uploaded weights may not be similar to each other. However, in the space of the whole dataset, the loss function should have a decreasing trend for those uploaded weights. Because the reference dataset is a subspace of the whole dataset, it can recognize this trend by examining the loss. Therefore, the reference dataset is an effective way to find how the uploaded weights perform and distinguish computational results between Byzantine nodes and honest nodes. In order to examine the performance of different nodes, the PS firstly computes the loss l_i based on the reference dataset. Then it normalizes l_i and filters out the weights with relatively larger loss using a loss filter parameter τ. The intuition behind this filter comes from Fig. 1 in [16] and Fig. 2 in [22], that is, Byzantine nodes must perform very badly in order to successfully attack the training process and therefore, the loss of the reference dataset must be much larger than those honest nodes. Through the normalization and filter process, it is easy to remove those outlier weights.

Algorithm 2 ToFi Algorithm (PS Side)

Input:

Weights computed from $worker_1$ to $worker_m$: $G_w = \{\overline{w_{t+1}^1}, \overline{w_{t+1}^2}, \cdots, \overline{w_{t+1}^m}\}$;
Weights at time $t, t-1$: w_t, w_{t-1};
Learning rate at time $t, t-1$: γ_t, γ_{t-1};
Reference dataset ξ_R;
Predefined loss filter parameter τ;
Predefined similarity filter parameter ζ.

Output:

Weights at time $t+1$: w_{t+1}.

1: Examine the loss for each worker with reference dataset $l_i = f(\overline{w_{t+1}^i}, \xi_R), i = 1, 2, \cdots, m$;
2: Compute the mean $\mu = \frac{1}{m}\sum_{i=0}^{m} l_i$ and standard deviation $\sigma = \sqrt{\frac{\sum_{i=0}^{m}(l_i-\mu)^2}{m}}$ for l_i;
3: Compute the normalized loss $L_i = \frac{l_i-\mu}{\sigma}$;
4: Filter the uploaded weights with the normalized loss $G_f = \{i | e^{-L_i} > \tau\}$;
5: Filter G_f with the similarity of the previous gradient direction $G_s = \{i | i \in G_f, SIM((w_t - \overline{w_{t+1}^i})/\gamma_t, (w_{t-1} - w_t)/\gamma_{t-1}) < \zeta\}$;
6: Derive the α-weight aggregation parameters by $\alpha_i = \frac{e^{-L_i}}{\sum_{i \in G_s} e^{-L_i}}, i \in G_s$;
7: Update the weights on time t: $w_{t+1} = \sum_{i \in G_s} \alpha_i \overline{w_{t+1}^i}$;
8: Send back the updated weights w_{t+1} to each worker.

Update similarity-based filter. After the reference dataset-based loss filter, we filtered out the uploaded weights who perform badly on the reference dataset. However, in some extreme scenarios such as all the uploaded weights are from Byzantine nodes in one synchronization, the normalized loss may have similar bad performance. Although this scenario may occur with a very low probability in the real world, for example, 30 out of 100 nodes are Byzantine nodes and in each iteration, only 10 nodes are selected, then the probability of all selected nodes are from Byzantine is around 10^{-6}, this still is a concern to pollute the training process. Therefore we propose an update similarity-based filter to filter out the weights that change much more than the update in the previous iteration. We compare the similarity of the update in this iteration and the previous iteration and filter out those who change too significantly. This is reasonable because the updates in the training process usually have momentum, and thus the update change is mild. This filter helps to deal with the extreme scenario that all participating nodes are Byzantine nodes or some Byzantine nodes are not examined by the reference dataset. Here we let $SIM(x, y) = \arccos \frac{x \cdot y}{||x|| ||y||} + b \cdot ||x - y||$. This guarantees the angle and distance change of weights in adjacent iterations are bounded. Even if there are still weights from Byzantine nodes, the influence on the current training process is not significant.

After these two filters, the weights that have extreme values are filtered out. Then we aggregate the rest weights using an α-weighted average. Here we use the normalized loss on the reference dataset to measure the contribution of different

nodes and set softmax of them as the α-weight. This helps the training process learn more information from all the uploaded weights in the heterogeneous federated learning environment. In the next section, we also show ToFi performs well in experiments with practical conditions.

5.3 Remarks and Comparisons

The naive algorithm actually uses a lot of information provided by the reference dataset. Therefore the performance depends on how good the reference dataset is. In fact, this algorithm is like searching the minimum point of a function in a hyperplane whose coordinates are $\overline{w_{t+1}^i}$ and the objective function is the loss function on the reference dataset. Thus the discrepancy between the loss function on the reference dataset and the real whole dataset decides the performance. In a word, the naive algorithm uses the reference dataset to examine the performance of computation results from each node's local dataset and search for the best next move by the information from both the local dataset and the reference dataset.

Compared to the naive algorithm, the performance of ToFi does not fully depend on the quality of the reference dataset. ToFi adopts two filters. The first filter of ToFi is actually a practical modification of the naive algorithm. ToFi uses this filter to clean out the weights with abnormal loss on the reference dataset, it is a better and faster way to compute the optimization function in the naive algorithm. In addition, ToFi adds a second similarity filter to filter out the abnormal weights whose change is too significant than before. These two filters can cooperate to filter out those attack information. The reference dataset in ToFi is more like an examination dataset to remove the outlier weights rather than a decisive dataset to decide how to aggregate the weights in the naive algorithm.

6 Experiments

In this section, several experiments are conducted to show the effectiveness of our algorithm. Because the naive algorithm is slow and inefficient in practice, we focus our experiments on ToFi.

6.1 Experiment Environments Setting

In order to show how those two major differences (non-i.i.d. dataset and the possible minority of honest nodes) affect the Byzantine problems, we first compare different algorithms with those two assumptions in the federated learning environment with all node participation (i.e., all nodes are selected to perform the computation) and in the end, we show how different algorithms perform in the federated learning environment with partial node participation (i.e., some nodes are selected to perform the computation).

In this experiment, there are two challenges to set up experiment environments. One is how to simulate a heterogeneous non-i.i.d. environment for our experiment dataset. The other is how to simulate the Byzantine attacks. In order to simulate a heterogeneous non-i.i.d. distributed environment, we conduct two different level methods. The first method is a naive heterogeneous environment. It is simulated by evenly dividing the whole dataset into slices horizontally and each worker keeps one slice subdataset. Assume we have n workers and the whole dataset is $\xi = \{x_1, x_2, \cdots, x_p\}$, then each worker keeps their subdataset as $\xi_i = \{x_{p//n\cdot(i-1)+1}, \cdots x_{p//n\cdot i}\}$. The other method is an enhanced heterogeneous environment. Similar to the previous method, we first sort ξ by the label. In this case, each subdataset only keeps the data with a similar label. The difference between ξ_i should be very significant. As for Byzantine attacks, in our experiment, we conduct three different types of attacks. The first is the Gaussian attack. We simply generate Gaussian noise as attack weights. The second method is wrong labeled attack [12,25]. We let the label of the Byzantine workers' data be randomly placed, then the Byzantine worker just normally computes the weights, and upload results with wrong labeled training results to the PS. The last method is one bit attack. For the uploaded weights, we only change one dimension of it with a random value. Although our algorithm is capable of defending Byzantine attacks in the bootstrap scenario, for simplicity, we fixed the Byzantine workers in the experiment. In fact, the bootstrap scenario should have a better performance because no subdataset is hidden by the Byzantine workers.

In our experiments, we use two most common datasets in Byzantine tolerant distributed machine learning area: MNIST dataset [23] and CIFAR-10 dataset [8]. Both datasets have 10 categories of labels. We use a server with 4 Nvidia GeForce GTX 1080Ti GPUs to simulate our experiments. For the federated learning environment with all node participation, we deploy 8 workers for the CIFAR dataset. Each GPU keeps 2 workers. For the federated learning environment, we deploy 100 workers, and 10 workers are randomly selected in each iteration. In our experiments, we will compare our α-weighted based algorithm with four other algorithms: (i) simply average aggregation with filtering out all Byzantine faults; (ii) score based algorithm Krum [2]; (iii) median-based algorithm GeoMedian [3]; (iv) distance-based algorithm FABA [16]. In the federated learning environment with partial node participation, we also compare with another reference dataset-based algorithm Zeno [22]. For ToFi, we set the loss filter parameter $\tau = 0.8$, similarity filter parameter $\zeta = 2, b = 1$ and randomly select 500 data from the test dataset as reference dataset and leave the rest as test dataset. We run all 5 algorithms with different Byzantine environments and Byzantine attacks described above.

6.2 Federated Learning with All Node Participation

We only show the results of CIFAR-10 here. The results of MNIST are presented in the Appendix.

Naive Heterogeneous Environment. We first compare ToFi with three classic methods and ground truth (filter out all Byzantine attacks, average aggregation) in the three Byzantine environments we described above. We choose ResNet-18 [5] as our neural network, 0.001 as the learning rate, and SGD with momentum and weight decay as optimizers. We use 10 as interval length to simulate a more general scenario. Because we just want to compare the effectiveness of defending against attacks, we do not optimize for the best accuracy. We first run experiments on naive heterogeneous environments along with no Byzantine environment, the results are in Fig. 2. As we can see from Fig. 2, Krum performs badly on defending against Byzantine attacks in naive heterogeneous environments. ToFi and FABA have good and stable performance on Gaussian attack and wrong label attack, but ToFi beats FABA on one bit attack. This is because in the naive heterogeneous environment, although the dataset is partitioned into slices, it is still randomly assigned. GeoMedian beats ground truth for some epochs, but the performance is really unstable in some epochs. As for no Byzantine environment, all methods perform well except Krum.

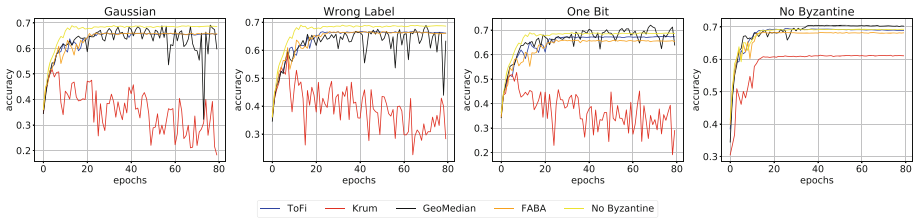

Fig. 2. Experiment results of different algorithms for Gaussian, wrong label, one bit Byzantine attacks and no Byzantine attack scenario on naive heterogeneous environment

Enhanced Heterogeneous Environment. We then compare different algorithms in the enhanced heterogeneous environment. Because the subdataset in each worker is very different and any complicated neural networks with batch normalization do not perform well in this environment, we use a LeNet [23] in this scenario. Using group normalization [15] to substitute batch normalization in complicated neural networks can be a good solution, but as we said that we only want to compare the performance of defending against attacks rather than chasing a good accuracy, we choose to use a simple neural network. The results are in Fig. 3. From Fig. 3, we can see that only ToFi and FABA are capable of defending against Byzantine attacks in this scenario, among which ToFi performs slightly better than FABA in the Byzantine environment and FABA is a little bit better than ToFi in no Byzantine environment. This is because that, in no Byzantine environment, the losses for all nodes are similar, so ToFi may filter out some useful information by both filters. Krum and GeoMedian have really

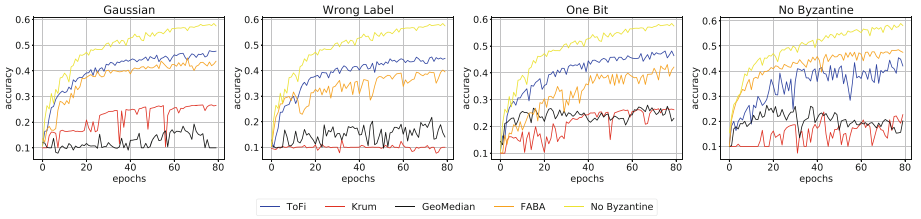

Fig. 3. Experiment results of different algorithms for Gaussian, wrong label, one bit Byzantine attacks and no Byzantine attack scenario on enhanced heterogeneous environment

bad performance in this environment. We can see that all algorithms do not perform very well compared to the ground truth, it is because, in this environment, a Byzantine attack may hide some labels of data such that the training data is missed and it will affect the performance.

Majority Attack. In order to simulate the majority attack, we choose 5 of 8 workers as Byzantine workers and use the same settings with the naive heterogeneous environment. Here we only examine the Gaussian attack because the other two types of attacks have similar performance. The results are in Fig. 4. It is obvious that Krum, GeoMedian, and FABA cannot defend against majority attacks. This is easy to understand because they only use information from uploaded weights. When the majority of workers are malicious, those methods will only take the malicious information as honest information and the whole training process will be dominated by Byzantine workers. ToFi uses the information from the reference dataset, so it is still able to defend against this kind of attack.

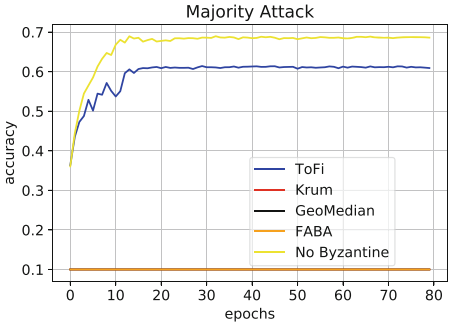

Fig. 4. Experiment results of majority Gaussian attack on naive heterogeneous environment

6.3 Federated Learning with Partial Node Participation

We compare Krum, GeoMedian, FABA, and Zeno with ToFi in the federated learning environment. In our setting, 30% of 100 nodes are Byzantine nodes and in each iteration, 10 nodes are randomly selected for computing. Here we use CIFAR-10 dataset, Gaussian attack, and enhanced heterogeneous environment. For other Byzantine attack types, the performance is similar. For FABA and Zeno that need to estimate the number of Byzantine nodes, we set it as $30\% \times 10 = 3$ in each iteration. The results are in Fig. 5. We can see that ToFi outperforms all other algorithms. It is because those algorithms are not designed to solve the two major differences in federated learning. When the distribution of each node's dataset is not i.i.d., Byzantine nodes may take the majority in some iterations, and the number of Byzantine nodes changes during the training, the performance downgrades a lot.

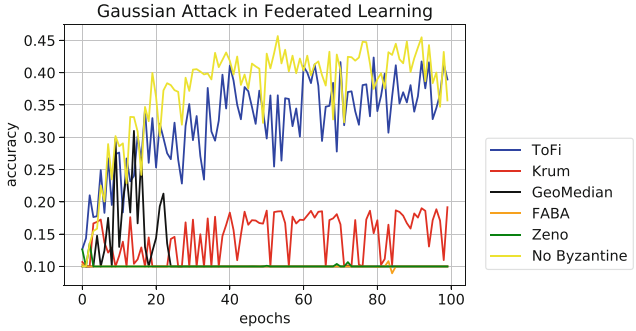

Fig. 5. Experiment results of Gaussian attack on enhanced heterogeneous environment in federated learning

7 Conclusion

In this paper, we address a practical issue of Byzantine attacks in federated learning. This is very different from Byzantine problems in classic distributed machine learning because, in federated learning, the distribution of the dataset in each worker is non-i.i.d., Byzantine workers can be the majority and the number of Byzantine nodes can change in different iterations. As far as we know, our paper is the first work to address this problem. We propose a naive algorithm and modify it to a reference dataset-based two-filter algorithm ToFi to adaptively filter out outlier computational results and take an α-weighted average based on the loss of all the uploaded weights from all involved workers as the aggregation results in each iteration. In our experiments, we compare ToFi with four existing algorithms in different environments with different kinds of Byzantine attacks and show the performance of our algorithm.

Acknowledgements. We thank all the reviewers for their constructive comments. 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.

A More Experiments on MNIST

All of the experiments in this section have the same setting as the main paper. We will mark any differences if there are. In this section, we supplement some results of experiments on the MNIST dataset and more workers [23]. The model we are using is LeNet-5 [23].

A.1 Federated Learning with All Node Participation

Naive Heterogeneous Environment. We compare our ToFi with three classic methods and ground truth (filter out all Byzantine attacks, average aggregation) in three Byzantine environments (Gaussian, wrong label, and one bit) we described in the main paper and no Byzantine environment. The distributed environment that we use here is the naive heterogeneous environment. We use 10 as interval length. The results are in Fig. 6. From this figure, we can see that the performance of Krum is not as good as ToFi, GeoMedian, and FABA, while these three methods have very similar performance in the naive heterogeneous environment for these three different types of attacks. As for the no Byzantine scenario, the performances are similar among ToFi, GeoMedian, and FABA, while Krum has a lower accuracy than those algorithms.

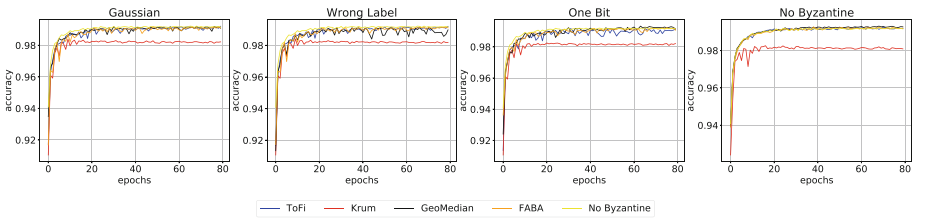

Fig. 6. Experiment results of different algorithms for Gaussian, wrong label, one bit Byzantine attacks and no Byzantine attack scenario on naive heterogeneous environment for MNIST dataset

Enhanced Heterogeneous Environment. In order to show the difference, we compare ToFi with three classic methods and ground truth (filter out all Byzantine attacks, average aggregation) in three Byzantine environments (Gaussian, wrong label, and one bit) and no Byzantine environment. This time we change the distributed environment to the enhanced heterogeneous environment. We use 10 as interval length. The results are in Fig. 7. From this figure, we can see that

ToFi has much better performance than Krum, FABA, and GeoMedian. GeoMedian has the second-best performance for Gaussian and one bit attacks. FABA has the second-best performance for wrong label attacks and no Byzantine scenario. But both of them have a significant accuracy decline than our algorithm. Krum has the worst performance in the enhanced heterogeneous environment.

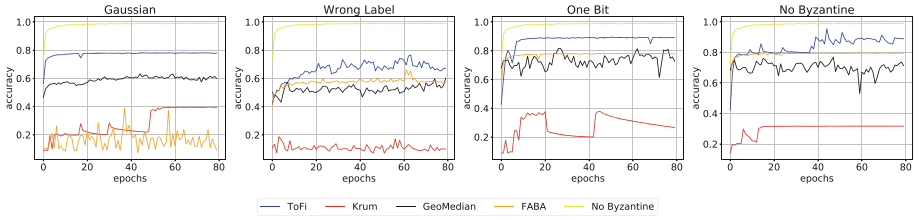

Fig. 7. Experiment results of different algorithms for Gaussian, wrong label, one bit Byzantine attacks and no Byzantine attack scenario on enhanced heterogeneous environment for MNIST dataset

More Workers Experiment. Because of the limitation of the hardware, we cannot make experiments for more workers than 8 on the CIFAR-10 dataset. Here we only examine the scenario with more workers on the MNIST dataset. In this experiment, we choose 32 workers, among which 8 out of 32 workers are Byzantine workers. To show the difference, we examine this setting in the enhanced heterogeneous environment. The results are in Fig. 8. From Fig. 8, it has a very similar performance with the 8-worker scenario. ToFi still outperforms other algorithms. For the Gaussian attack, ToFi has a similar performance with FABA and beats all other algorithms. For wrong label attack and one bit attack, ToFi performs much better than others. The best performance here is not as good as no Byzantine attack case. It is because in the experiment we fixed the workers who suffer Byzantine attacks. Since in this experiment we use the enhanced heterogeneous environment, the data with some labels may be hidden by the Byzantine workers. This will cause a decrease in the accuracy for the best performance.

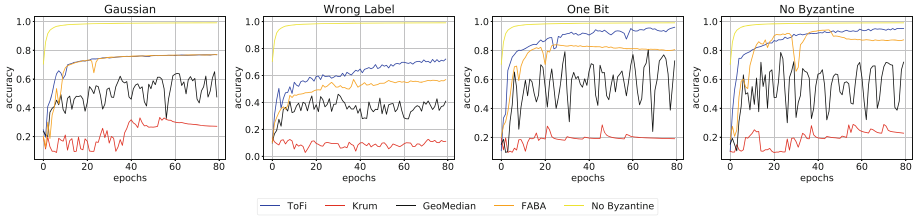

Fig. 8. Experiment results of different algorithms for Gaussian, wrong label and one bit Byzantine attacks on enhanced heterogeneous environment with 32 workers for MNIST dataset

A.2 Federated Learning with Partial Node Participation

We compare Krum, GeoMedian, FABA and Zeno with ToFi in the federated learning environment using similar setting with CIFAR-10 dataset. The results are in Fig. 9. We can see that ToFi outperforms all other algorithms. All the other algorithms are not designed for federated learning and thus have very bad performance.

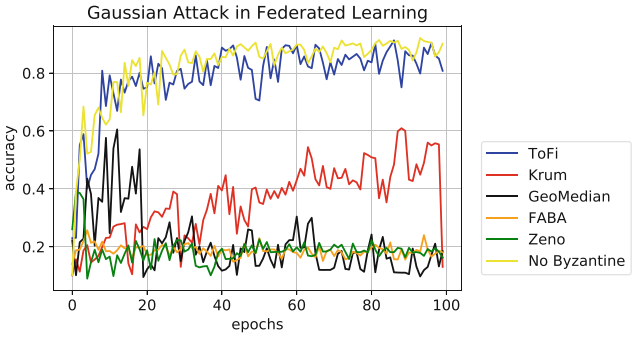

Fig. 9. Experiment results of Gaussian attack on enhanced heterogeneous environment in federated learning

References

1. Alistarh, D., Allen-Zhu, Z., Li, J.: Byzantine stochastic gradient descent. CoRR abs/1803.08917 (2018). http://arxiv.org/abs/1803.08917
2. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: byzantine tolerant gradient descent. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30, pp. 119–129. Curran Associates, Inc. (2017). http://papers.nips.cc/paper/6617-machine-learning-with-adversaries-byzantine-tolerant-gradient-descent.pdf
3. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proc. ACM Meas. Anal. Comput. Syst. **1**(2), 1–25 (2017). https://doi.org/10.1145/3154503
4. Damaskinos, G., El Mhamdi, E.M., Guerraoui, R., Patra, R., Taziki, M.: Asynchronous Byzantine machine learning (the case of SGD). In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1145–1154. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018). http://proceedings.mlr.press/v80/damaskinos18a.html
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
6. Konecný, J., McMahan, H., Yu, F., Richtárik, P., Suresh, A., Bacon, D.: Federated learning: strategies for improving communication efficiency. CoRR abs/1610.05492 (2016)

7. Konstantinov, N., Lampert, C.: Robust learning from untrusted sources. CoRR abs/1901.10310 (2019). http://arxiv.org/abs/1901.10310
8. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report, Google (2009)
9. Mao, Y., Hong, W., Wang, H., Li, Q., Zhong, S.: Privacy-preserving computation offloading for parallel deep neural networks training. IEEE Trans. Parallel Distrib. Syst. **32**(7), 1777–1788 (2021). https://doi.org/10.1109/TPDS.2020.3040734
10. Mao, Y., Yi, S., Li, Q., Feng, J., Xu, F., Zhong, S.: Learning from differentially private neural activations with edge computing. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 90–102 (2018). https://doi.org/10.1109/SEC.2018.00014
11. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) (2017). http://arxiv.org/abs/1602.05629
12. Paudice, A., Muñoz-González, L., Lupu, E.C.: Label sanitization against label flipping poisoning attacks. In: Alzate, C., et al. (eds.) ECML PKDD 2018 Workshops, pp. 5–15. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13453-2_1
13. Tao, Z., Li, Q.: eSGD: Communication efficient distributed deep learning on the edge. In: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18). USENIX Association, Boston, MA, July 2018
14. Tao, Z., et al.: A survey of virtual machine management in edge computing. Proc. IEEE **107**(8), 1482–1499 (2019). https://doi.org/10.1109/JPROC.2019.2927919
15. Wu, Y., He, K.: Group normalization. Int. J. Comput. Vis. **128**(3), 742–755 (2020). https://doi.org/10.1007/s11263-019-01198-w
16. Xia, Q., Tao, Z., Hao, Z., Li, Q.: FABA: an algorithm for fast aggregation against byzantine attacks in distributed neural networks. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pp. 4824–4830. International Joint Conferences on Artificial Intelligence Organization (July 2019). https://doi.org/10.24963/ijcai.2019/670
17. Xia, Q., Tao, Z., Li, Q.: Defenses against byzantine attacks in distributed deep neural networks. IEEE Transactions on Network Science and Engineering (2020). https://doi.org/10.1109/TNSE.2020.3035112
18. Xia, Q., Tao, Z., Li, Q.: Privacy issues in edge computing. In: Chang, W., Wu, J. (eds.) Fog/Edge Computing For Security, Privacy, and Applications. AIS, vol. 83, pp. 147–169. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-57328-7_6
19. Xia, Q., Ye, W., Tao, Z., Wu, J., Li, Q.: A survey of federated learning for edge computing: Research problems and solutions. High-Confidence Computing (2021). https://doi.org/10.1016/j.hcc.2021.100008
20. Xie, C., Koyejo, O., Gupta, I.: Generalized byzantine-tolerant SGD. CoRR abs/1802.10116 (2018). http://arxiv.org/abs/1802.10116
21. Xie, C., Koyejo, O., Gupta, I.: Zeno++: Robust fully asynchronous sgd (2020). https://openreview.net/forum?id=rygHe64FDS
22. Xie, C., Koyejo, S., Gupta, I.: Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 6893–6901. PMLR, Long Beach, California, USA (09–15 Jun 2019). http://proceedings.mlr.press/v97/xie19b.html
23. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791

24. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: towards optimal statistical rates. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 5650–5659. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018). http://proceedings.mlr.press/v80/yin18a.html
25. Zhang, M., Hu, L., Shi, C., Wang, X.: Adversarial label-flipping attack and defense for graph neural networks. In: 2020 IEEE International Conference on Data Mining (ICDM), pp. 791–800 (2020). https://doi.org/10.1109/ICDM50108.2020.00088