2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Metaverse)

Attack-Model-Agnostic Defense Against Model Poisonings in Distributed Learning

Hairuo Xu and Tao Shu
Department of Computer Science and Software Engineering
Auburn University
Auburn, AL, USA
{hairuoxu, tshu}@auburn.edu

***Abstract*—The distributed nature of distributed learning renders the learning process susceptible to model poisoning attacks. Most existing countermeasures are designed based on a presumed attack model, and can only perform under the presumed attack model. However, in reality a distributed learning system typically does not have the luxury of knowing the attack model it is going to be actually facing in its operation when the learning system is deployed, thus constituting a *zero-day vulnerability* of the system that has been largely overlooked so far. In this paper, we study the attack-model-agnostic defense mechanisms for distributed learning, which are capable of countering a *wide-spectrum* of model poisoning attacks without relying on assumptions of the specific attack model, and hence alleviating the zero-day vulnerability of the system. Extensive experiments are performed to verify the effectiveness of the proposed defense.**

***Index Terms*—distributed machine learning, attack-model-agnostic defense, heritage factor, threat and detection model**

I. Introduction

In recent years we have witnessed the initial success of machine-learning-based AI (artificial intelligence) in many application domains, such as image processing, natural language processing, autonomous driving, robotics, gaming, medical science, and public safety. Encouraged by this initial success, the size of learning is being scaled up, so as to make a trained model more generalizable and applicable to wider scopes. This is achieved by increasing not only the total volume of data used to train the model, but also the number of independent sources that collect the data under different spatial and temporal scenarios and contribute them for model training. For example, the medical imaging records from multiple independent medical institutions can be garnered to train a disease diagnosis model that is more accurate than what can be achieved by using data from any one institution alone [1]. In line with this momentum, distributed machine learning technology has received a lot of interests recently. A distributed machine learning algorithm allows each source to first train an individual model (a.k.a. child model) just based on its own dataset, and then utilize the training result from all child models to construct a generalized model. As such, distributed machine learning enjoys the highly desirable benefit of data privacy, because it only requires the sharing of child model's training outcome (i.e., training parameters or gradients), rather than a direct disclosure of data across different sources. In addition, the training over different data sources can be paralleled and crowd-sourced to a cluster of machines, and therefore parallelism and high-speed is another benefit provided by distributed machine learning.

While distributed machine learning provides many nice features, researchers are concerned about its security problems, especially its vulnerability to model poisoning attacks. In particular, because the validity of the final learning outcome depends on the correctness of every child model and as the learning has a distributed structure, an attacker may simply compromise a subset of the data sources and tamper the training of their child models to compromise the final generalized model. Similar damage may also be caused when some data sources are malicious, injecting false child model parameters (or gradients) into the distributed learning process. Even worse, the iterative structure of many distributed machine learning algorithms allow the injected false model parameters to propagate among both global and child models, eventually affecting the validity of both the generalized model and all child models. In light of these threats, the countermeasures that allow the model server to check the validity (or the likelihood of being valid) of the shared local training outcome, and to eliminate the suspicious workers is extremely important, as it provides the guarantee for the convergence and the quality of the final global model in the distributed learning process.

Most existing countermeasures in distributed learning are designed based on a presumed attack model, and present their defense capabilities under that presumed attack model. However, they achieve limited defense effectiveness when facing different attack models other than the one that it was designed against. For example, the defense methods described in [2] and [3] are focusing on the targeted attack models, and their common goal is to prevent the mis-classification of a certain label. However, when facing the untargeted attack which aims to distort the global convergence, such targeted attack defense methods very often achieve poor performance.

In reality, however, when a distributed learning system is being deployed, it typically does not have the luxury of knowing the attack models that will actually be launched against it in its operation, thus constituting a *zero-day vulnerability* for the distributed learning system. Pre-stacking up a variety of countermeasures during the deployment of the

979-8-3503-4655-8/22/$31.00 ©2022 IEEE

DOI 10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00354

learning system, each of which is designed specifically for a particular attack model that is anticipated to occur to the underlying system - a method commonly adopted by the software engineering industry to counter viruses, is not a feasible solution for learning systems because the countermeasures against different attack models are typically not compatible with each other. Therefore, when a new distributed learning system is being deployed, it is critical to embed in the system a *wide-spectrum* counter-attack defense mechanism, whose operation relies on little assumptions of the attack model (i.e., being attack model agnostic), and thus is able to counter a wide range of possible attacks. Such a defense mechanism will give the learning system an effective first line of defense when the system is born, alleviating the zero-day vulnerability of the system.

In this paper, focusing on the general category of model poisoning attacks, we study the attack-model-agnostic defense mechanisms in distributed machine learning by exploring several main-stream untrageted attack countermeasures such as Krum [4] and AFA [5]. We also propose two new attack-model-agnostic countermeasures to defend against general model poisoning attacks in distributed learning: the Drop-one, and Structural Similarity (SSIM) detection. We evaluate these detection methods against several main stream untargeted attack models, and compare their effectiveness under different attack settings. Our contributions include the following three folds:

- Targeting the general category of model poisoning attacks, we propose two novel attack-model-agnostic defense methods for distributed learning: Drop-One and SSIM detection. It is worth noting that in contrast to the existing detection methods that compute the decision boundary based on the snapshot of the training outcome in current iteration, the SSIM method considers the trace of training outcomes in a sequence of iterations (i.e., the current and the past K iterations). For better detection accuracy, SSIM regards the gradients from each epoch as a series of vectors, and finds out the most suspicious gradients by exploiting historic information.
- We propose a new metric, the Heritage Factor, to measure and study the false parameter propagation between child models in different iterations of distributed learning. Such a metric also enables us to characterize the impact of attack injected initially into a single child model on the validity of the final generalized model.
- We conduct extensive model poisoning attack experiments over the MNIST dataset under a wide range of attack models, and observe that the magnitude of the injected attack vector is the dominant factor for the effectiveness of the attack regardless of the attack model. Based on this observation, we verify through extensive experiments the effectiveness of the proposed attack-model-agnostic countermeasure mechanisms over a wide range of attack magnitudes.

To the best of our knowledge, our work is the first to systematically study the attack-model-agnostic countermeasures against model poisonings in distributed learning systems.

II. Related Work

A. Poisoning Attacks

Distributed learning is vulnerable to poisoning attacks. According to the goal of the attackers, the poisoning attacks can be divided into two categories, the targeted attacks that aims to reduce the accuracy on a given class, as described in [6] and [7], and the untargeted attack, whose goal is to prevent the distributed learning system from converging to its optimal solution as described in [8] [9] [10] [11]. We believe that the untargeted attacks that threat the entire learning system has severe consequences than targeted attacks, therefore, we focus on the defense mechanisms against the untargeted attacks in this paper.

B. Defense Methods

Based on the potential attack models, the defense method could also be divided into two categories. The targeted defense proposed in [2] and [3] were based on the prior knowledge of the attack models. And more generally, Krum and Multi-Krum [4], AFA [5], Trimmed-Mean [10], Median [12] and Bulyan [13] are proposed to counter the untargeted attacks in distributed learning systems. These countermeasures utilize the statistical information such as gaussian similarity, cosine similarity, mean and median along each dimensions of the uploaded gradients, and draw the decision boundary to eliminate the outlying votes which are regarded as the attack vectors. It is worth noting that most of these methods compute the decision boundary based on the snapshot of the gradients in the current iteration, the SSIM proposed in this paper utilize the current and the historical gradients to differentiate the suspicious collaborators from the honest ones.

III. Model Description and Problem Formulation

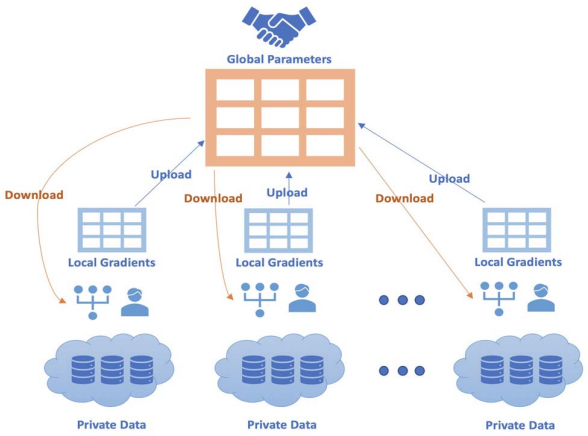

Fig. 1. High-level system architecture.

A. Overview

Figure 1 illustrates the high-level architecture of our distributed learning system. It is an abstract example of the parameter server [14] system. The system contains one server, which is responsible for maintaining the global parameters, and N collaborators (also referred to as peers in the following text). There exists the communication protocol that allows the collaborators to share the training information through local gradients upload and global parameters download. In the following, we describe the role and tasks of each components: the parameter server and the collaborators in details.

B. Local training

Assume that there are N peers, all of which have agreed in advance upon the same learning objective and training model architecture. For simplicity, but without loss of the generality, we choose the classification task as our objective, and neural network as our training architecture in this paper. Each peer maintains their own private dataset. These dataset should not be shared as they may contain sensitive information.

Each peer i maintains a local set of neural network parameters, denotes to $\mathbf{w}_k^i$. The peer starts training by downloading, and replacing all local parameters with the global parameters from the server. After that, the peer trains locally for one epoch with Stochastic Gradient Descent optimization [15]. In this epoch, the peer should train on all the local data, instead of certain mini-batches. This single-epoch training produces a set of gradients, denotes to $\mathbf{g}_k^i$ as

$$\mathbf{g}_k^i = \mathbf{w}_k^i - \mathbf{w}_{k-1}^i \tag{1}$$

where

- $\mathbf{g}_k^i$ denotes to peer i's true local gradients trained with SGD at epoch k.
- $\mathbf{w}_k^i$ denotes to the local weights(parameters) of peer i at epoch k after the local training.

The peers will then upload the gradients $\mathbf{g}_k^i$ to the parameter server, and wait until the next iteration.

The next peer will perform the same process of downloading and replacing the local parameters by the parameters from the server, which now contains the information uploaded by the previous peer, train locally for one epoch, stop and upload the gradients to the server. The iteration continues until all the peers have trained locally for one epoch and then the next iteration starts, where each peers will train for the second epoch.

C. Parameter server

The parameter server maintains a set of global parameters, denotes to $\mathbf{w}_k^{\text{global}}$. Each time the server receives a gradient vector from a peer, it updates the set of global parameters by the following formula:

$$\mathbf{w}_k^{\text{global}} = \mathbf{w}_{k-1}^{\text{global}} + \eta \mathbf{g}_k^i \tag{2}$$

where

- η is a constant that characterizes the percentage of uploaded gradient to be aggregated into the global parameters (or learning rate in conventional machine learning).
- $k-1$ and k denote the epoch number, and
- $\mathbf{g}_k^i$ is the local gradients upload by peer i at epoch k.

As shown in Figure 1, each upload is followed by an update, and an iteration is finished after each peer has trained itself for one epoch and has uploaded its local gradients.

IV. Threat Models

Our proposed defense mechanisms seek to counter a full range of model-poisoning attacks in distributed learning without requiring pre-knowledge of the specific attack model. Several representative model-poisoning attacks are presented below. Note that our proposed defense mechanisms do not depend on any of these specific attack models. Instead, as will be clear shortly in Section VI, we will make observation on some common behavior presented by these attack models, which justifies our proposed model-agnostic defense mechanisms.

A. Single Attacker

Similar to the honest peers, the attacker has access to its own private data, and the communication channel with the server. In order to take advantage of other collaborators' training efforts, the attacker download the global parameters once. That's the only time that the attacker will perform parameters downloading, and that particular set of global parameters downloaded will be the last set of "pure" global parameters, as it contains the training information from the honest peers and has not been polluted yet.

After the initial download of global parameters, the attacker solely trains on its own data. When it's the attacker's turn to upload, the attacker fabricates a vector resembling the features of the true gradients, and uploads it to the server. We consider the following two model poisoning attack models, and explain their strategies to fabricate the poisoning vector with the maximum attacking effects.

1) Random Noise Attack: The random noise attack fabricates the poisoning vector by uniformly generating a set of random scalars within the range (also referred to as magnitude in the following text) of $[\tilde{r_1}, \tilde{r_2}]$ as

$$\widetilde{\mathbf{g}_k^i} = \boldsymbol{\epsilon} \sim \text{unif}(\tilde{r_1}, \tilde{r_2}). \tag{3}$$

where $\boldsymbol{\epsilon}$ is uploaded to the parameter server as the true gradients. In experiments, we create several groups of $[\tilde{r_1}, \tilde{r_2}]$ in order to observe the correlation between the poisoning magnitude and the attack strength. At the same time, the attacker takes the advantage of the global parameters downloaded, and trains itself for one epoch.

2) Gradient Ascent Attack: Gradient ascent attack fabricates the poisoning vector by first creating a temporary model. Then, the attacker downloads the global parameters into the temporary model, and perform **gradient ascent**

optimization for one epoch. The resulting gradients, serving as the poisoning vector, will be uploaded to the server. Similar to Equation 1, the gradients is computed by subtracting the parameters downloaded from the parameters after gradient ascent training in current epoch as

$$\widetilde{\mathbf{g}_k^i} = \boldsymbol{\epsilon} = \mathbf{w}_k^i - \mathbf{w}_{k-1}^i \tag{4}$$

where $\mathbf{w}_{k-1}^i$ is equivalent to the global parameters $\mathbf{w}_{k-1}^{\text{global}}$ which was downloaded to the temporary model.

It is worth noting is that the ascending algorithms is based on the global parameters, not the attacker's local parameters, as the local parameters only contain the converging information of the attacker's data, and at the server-point-of-view, it may not be the steepest ascending direction. The global parameters, instead, contain the converging information from all the honest peers, and the ascending gradients computed from these parameters should yield to a better attacking performance.

In order to compare the attacking impacts with the random noise attack, we also take the magnitude of the ascent vector into consideration. After the computation of $\boldsymbol{\epsilon}$ using gradient ascent, it is normalized into different scale ranges. And in general, the gradient ascent attack can be considered as a more optimized case of the random noise attack, which the injected random attack vector is the carefully computed ascending gradients.

B. Multiple Attackers

In this subsection, we inject the distributed attacking flavor into the previously-described single-attacker models. It is assumed that all the attackers know the identity of other attackers and can share information with others under the condition of not revealing their own data.

1) Collaborative Random Noise Attack: Compared to the random noise attack in the single-attacker scenario, collaborative random noise attack can be seen as multiple attackers working together towards the same attacking goal. With a set overall magnitude (or range) of the attack vectors, the group of attackers will collaborate to produce multiple attack vectors, pretending to be the true gradients.

Denote the magnitude (range) of the overall attack vector as $[\tilde{r_1}, \tilde{r_2}]$, and denote the number of attackers as N_a, The attack vector that each attacker will generate is:

$$\widetilde{\mathbf{g}_k^i} = \boldsymbol{\epsilon} \sim \text{unif}(\frac{1}{N_a}\tilde{r_1}, \frac{1}{N_a}\tilde{r_2}) \tag{5}$$

Such collaborative attack strategy ensures the consistency of the overall attack magnitude, while assigning the task of computing a sub-attack vector to individual attackers.

2) Collaborative Gradient Ascent Attack: In Collaborative Gradient Ascent Attack, one attacker computes the global **ascending** gradients, upload part of the ascending gradients to the server, and send the rest to the next attacker. The next attacker will perform the same process of uploading another part of the remaining gradients and send the rest to the next attacker. Denote N_a to the number of attackers, and a_1, a_2,...,a_n to the attacker's IDs, respectively. The first attacker, a_1, computes the ascending gradients $\boldsymbol{\epsilon}$ according to Equation 4. The poisoning vectors that will be uploaded by each attacker is:

$$\widetilde{\mathbf{g}_k^{a_n}} = \frac{1}{N_a - (n-1)}(\boldsymbol{\epsilon} - \sum_{j=1}^{n-1}\widetilde{\mathbf{g}_k^{a_j}}) \tag{6}$$

C. Heritage Factor

With plenty of threat models explained, in this section, the essence of the attack models is described. We propose a new concept, the Heritage Factor, under this particular architecture of distributed learning system.

As already explained in Eq.(2), each collaborator is responsible for uploading its local training gradients, denotes to $\mathbf{g}_k^i$. For honest peers, $\mathbf{g}_k^i$ is the true gradients that is produced from local training at epoch k, however, for adversaries, the gradients that they upload contain poisoning vectors with different magnitudes.

It is worth noting that if there exists no attacker, the distributed learning system should help each of the collaborators to learn a more generalized model and converge faster [16], [17]. From a human's understandable perspective, we can regard such learning procedure as a inheritance behavior. The later peers "inherit" all the legacies from the previous collaborators. The "legacy" is contained in the true gradients that are aggregated by the update function on the parameter server.

From an attacker's perspective, the vectors uploaded can be seen as a weighted sum of the attackers' true gradients and the attack vector. Denote the uploaded vector from an attacker as $\widetilde{\mathbf{g}_k^i}$, we can formulate $\widetilde{\mathbf{g}_k^i}$ as:

$$\widetilde{\mathbf{g}_k^i} = \rho\mathbf{g}_k^i + (1-\rho)\boldsymbol{\epsilon} \tag{7}$$

where

- $\widetilde{\mathbf{g}_k^i}$ describes the vector to be uploaded by attacker i,
- $\mathbf{g}_k^i$ represents the true gradients produced by local training,
- $\boldsymbol{\epsilon}$ denotes the attack vector, and
- we define $0 \leq \rho \leq 1$ to be the **heritage factor**, which is a coefficient that specifies the percentage of the true gradients information that will be inherited.

When ρ is 1, no attack vector is added to the true local gradients, and the so-called attacker is not performing any attacking activities. On the other extreme, when ρ is 0, the entire uploaded vector would be the attack vector, and it will contain no learned information (or legacy) from the local model.

V. Defense Methods

In this section, we first describe the two proposed attack-model-agnostic defense models, and then summarize the

main-stream untargeted detection mechanisms. All the defense methods are developed to be applied before the aggregation of the local updates on the server to eliminate the suspicious collaborators (or attackers).

A. Drop One Detection

As the name suggest, the Drop One Detection method is applicable when there's only one attacker. We put the uploaded gradients into groups. Denote the set of collaborators as $\mathbb{C}$, where $|\mathbb{C}| = N$. We generate N groups, and each group contains $N-1$ uploaded gradients, that is, each group contains all but the gradients from the peer who has the same ID as the group ID. More specifically, group j contains the set of uploaded gradients $\{\mathbf{g}_k^i \mid \forall i \in \mathbb{C}, i \neq j\}$.

After the grouping, the standard deviation of each group is calculated, and compared with other groups'. The group, in which the standard deviation is the smallest, is regarded as the group in which the attacker's gradients vector is dropped.

B. Series Structural Similarity

The Structural Similarity (SSIM) was first proposed in [18] aiming to find the images that contain the best human perceptual information after the same amount of Mean Squared Error (MSE) distortion. From the perspective of computer vision, the Structural Similarity measures the luminance, contrast, and the temporal and spatial correlations of pixels in an image. In our scenario, comparing images is out of our scope, instead, our goal is to find out the outliers from a set of uploaded gradients vectors.

As the collaborators had all agreed on the local training architecture and the training objectives, if there exists no attacks, the gradients uploaded from each of the collaborators will contain the common converging information that points towards the global optimal solution. Therefore, by considering the uploaded gradients at each epoch as the structural information of the overall distributed learning system, we present the SSIM defense mechanism. Denote the series of uploaded vectors of peer i by $\mathbf{T}^i = \{\mathbf{g}_1^i, \mathbf{g}_2^i, ..., \mathbf{g}_k^i\}$, where the $1, 2, ..., k$ represents the epochs number. The SSIM mechanism computes the structural similarity score [18] to analyzes the trace of each peer and predicts the potential attackers. The intuition is that the attackers would leave a different trace in comparison with other honest collaborators.

A subtle change is made because of the fact that the ground truth (or the "original reference image" in computer vision) does not exist in the case of distributed learning, and therefore multiple references are randomly chosen and the SSIM scores are computed according to each reference. The majority vote is taken from all references and the suspicious collaborator(s) are eliminated.

C. Gaussian Detection (Krum)

The Krum detection method is proposed in [4] based on the assumption that all gradients should follow the Gaussian distribution, and that attackers/outliers should reside far away from honest collaborators. As previously denoted, the collection of all the uploaded vectors at epoch k is $[\mathbf{g}_k^1, \mathbf{g}_k^2, \ldots, \mathbf{g}_k^n]$. The gradients mean μ_k and the standard deviation σ_k are computed first. Then, for each uploaded vector, its distances to the mean is calculated, and is represented by the multiples of standard deviation σ_k as

$$\theta_k^i = (\mathbf{g}_k^i - \mu_k)/\sigma_k \tag{8}$$

A larger θ_k^i indicates a larger distance from the mean, and the corresponding uploader is more suspicious. Define the outlier threshold δ_{Gaussian} in epoch k to be the $(1-\alpha)$ percentile of the Gaussian distribution of zero mean and σ_k^2 variance, where α is a small and given probability that defines the outliers. Then if $\theta_k^i > \delta_{\text{Gaussian}}$, its uploader i is regarded as an attacker.

The Gaussian Detection method focuses more on the gradients vectors' magnitude, as the calculation of mean and standard deviation ignores the high-dimension direction of gradient vectors. The threshold used is observed from the experiments to obtain the best accuracy.

D. Cosine Similarity Detection (AFA)

AFA [5] considers the Cosine Similarity between the uploaded vectors. Instead of considering the magnitude of uploaded gradients like Krum, AFA is developed under the intuition that the attackers/outliers' high dimensional gradient direction are different from those from honest peers, which instead may be pointing towards the global convergence point. AFA starts by computing the mean μ of all the uploaded vectors. Then, the cosine similarity score between each uploaded vector and the mean are computed as

$$cos(\beta) = \frac{\mathbf{g}_k^i \cdot \mu}{||\mathbf{g}_k^i|| \cdot ||\mu||} = \frac{\sum_{j=1}^n \mathbf{g}_{k_j}^i \mu_j}{\sqrt{\sum_{j=1}^n \mathbf{g}_{k_j}^{i2}} \sqrt{\sum_{j=1}^n \mu_j^2}}. \tag{9}$$

A smaller cosine similarity score indicates the less similarity from the mean, and the corresponding collaborator who obtain a small score is more suspicious.

VI. Evaluation

A. Dataset and setup

We evaluate our threat and detection models using the MNIST dataset [19] which contains $60,000$ images as the training set, and $10,000$ in the testing set. We implement a distributed learning system which includes one parameter server and multiple collaborators, each maintain the same neural networks architecture. The training data are shuffled and separated randomly into each peer.

We use PyTorch [20], version $0.4.1$, with the help of Numpy and Torchvison packages, to implement our algorithms. We set the global and local learning rate η to 0.01, and batch size to 64. Cross Entropy is implemented as the loss function and Stochastic Gradient Descent is adopted as the optimizer. We conduct our experiments on Tesla P100 GPUs, and all of our test results shown below are the averages of 50 runs.

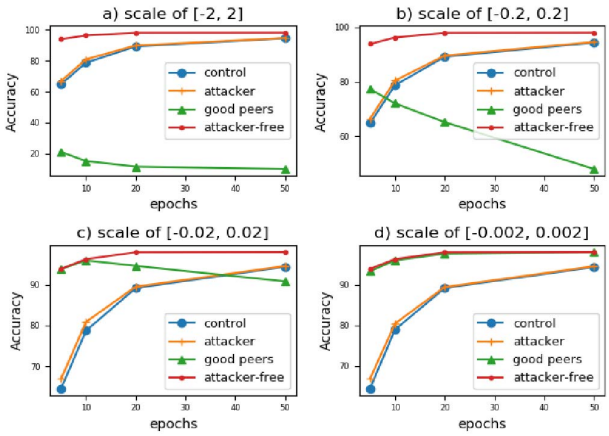

Fig. 2. Performance results of $N = 10$, single attacker, random noise attack with multiple magnitudes.

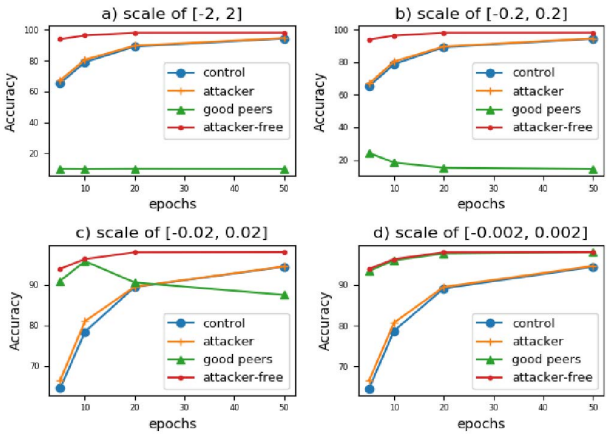

Fig. 3. Performance results of $N = 10$, single attacker, gradient ascent attack with multiple magnitudes.

B. Attacking Results

1) Single Random Noise Attack: Experiments are conducted simulating the scenario where among all the collaborators, only one of them is an attacker, and that attacker is launching the random noise attack against the distributed learning system. Figure 2 demonstrates the relationship between the magnitude of attack vectors versus the final classification accuracy.

The attacker-free and control curves correspond to the scenarios where there is no attacker, and when one peer does not participate in the distributed learning. From this figure, it is clear that the larger the magnitude of the attack vectors are, the bigger disturbance is injected into the learning system, therefore, the worse the classification performances are for the honest peers.

Another observation is that when the attack magnitudes are large (the first three subplots), the more epochs of training conducted, the more disturbance the system has. Our guess is that from the first several training epochs, the local training has not converged yet, so its uploaded gradients vectors do not contain enough converging information, causing the global parameters to be polluted by the non-converging attack vectors. The injected poisoning is then propagated to other collaborators by global parameters downloading, and spread throughout the entire distributed learning system, affecting both the global server and all collaborators. Although each honest collaborator is trying to minimize the loss, they are not able to get to the final near-convergence stage since they are only allowed to train for one epoch, which is not enough to correct the poisoned parameters. This vicious process goes on and on, and will not be reverted back. As a result, the more training the system performs, the further away it is from the optimal convergence point.

Another experiment is conducted to verify our guess. We implemented the defense methods described in section V to find out the potential attacker(s) after several epochs of training. Once a collaborator is classified as an attacker, the server will stop using its upload for the global parameters updating purposes. It turns out that at first, the distributed learning system is not converging at all, but when the server depicts and eliminates the uploaded vectors from the attacker, the system starts to converge, and the performance results of the honest collaborators starts to increase. Due to the space limitation, the resulting figure is not present in this paper.

Such observation also correlates with our definition of the Heritage Factor. The smaller the Heritage Factor is, the less percentage of true gradients is inherited, causing the overall performance to be degraded. The detailed experimental results for Heritage Factor is demonstrated later this section.

2) Single Gradient Ascent Attack: The experiments for single gradient ascent attack are launched under the same settings as the single random noise attack. Figure 3 demonstrates the performance results of among 10 collaborators, there exists one attacker and the Gradient Ascent Attack is launched. Similar to the ones in the single random noise attack, the following observations can be extracted:

- the bigger the magnitude of the attack vector is, the worse the performance of the honest collaborators are.
- the "diverging" information propagates within the system, degrading the performance with the increase number of training epochs.

Comparing the subplots from Figures 2 and 3, it is suggested that when the magnitude of the attack vector is set, the gradient ascent attack has better attacking effectiveness, as it drags the global parameters to the steepest ascending direction, whereas in the random noise attack, the global parameters are dragged to random directions. And again, in such perspective, the gradient ascent attack can be seen as a special case of the random noise attack.

3) Multiple Attackers: Figures 4 and 5 demonstrates the experiment results when there are 10 collaborators and 2, 4 of them are attackers, respectively. The attackers share the overall attack magnitude, and in this case, let the overall

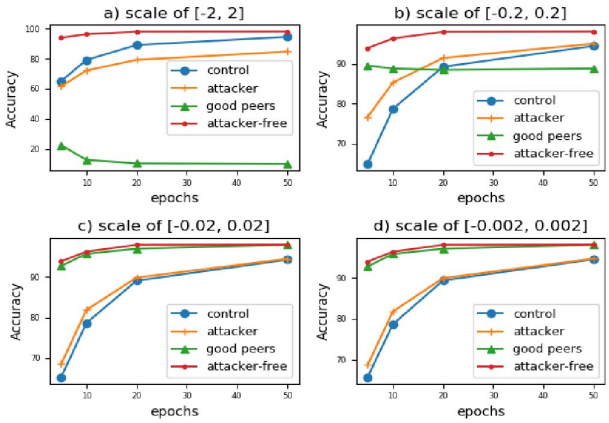

Fig. 4. Performance result of $N = 10$, 2 attackers, collaborative random noise attack.

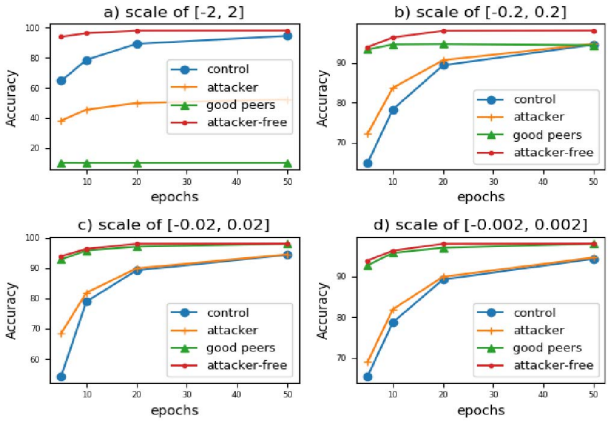

Fig. 5. Performance result of $N = 10$, 4 attackers, collaborative random noise attack.

attack magnitude be $[r1, r2]$, then the magnitude that each attacker leverages is $[\frac{1}{2}r1, \frac{1}{2}r2]$, and $[\frac{1}{4}r1, \frac{1}{4}r2]$, respectively. Comparing Figure 2, 4 and 5, where the only changing variable is the number of attackers (or the attack magnitude of each attacker), the performance of honest peers in Figure 2 is the lowest, followed by 4. This suggests that injecting a single large attack vector has a stronger attack strength on the distributed learning system than injecting small attack vectors multiple times. The experiment results for multiple Gradient Ascent Attack also suggest the same insights, the results are omitted due to the limited space of the paper.

C. Defense Results

From Figures 2-5, it can be observed that the magnitude of the injected attack vector is the dominant factor in deciding the ultimate attack effectiveness, regardless of the attack models. Therefore, in this subsection, we will focus on this dominant factor and perform our defense over a wide range of attack magnitudes. Ideally, we prefer to show our defense results against all attack models presented in Section IV. However, due to the space limit, here we can only present

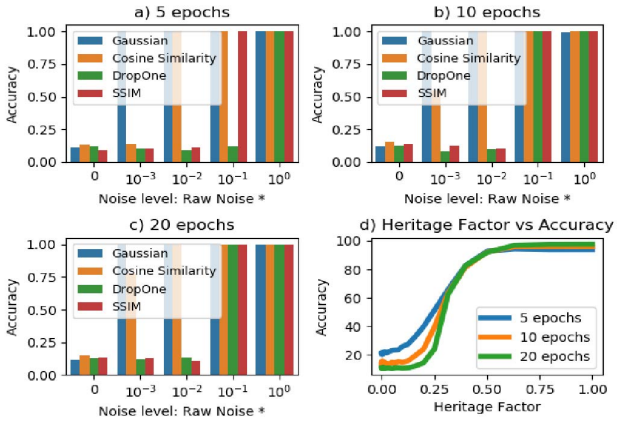

Fig. 6. a)-c) Performance result of $N = 10$, single attacker, random noise attack defense at multiple magnitudes. d) the correlation between heritage factor and Accuracy.

the performance of our proposed defense mechanisms under the random noise attack. This is because the random noise attack can be seen as the most general case of the untargeted attack, and other attack models can be regarded as special cases of the random noise attack. Therefore, the performance of the proposed defense mechanisms under the random noise attack would be representative.

1) Single Attacker Defense: Figure 6 a) - c) demonstrates the performance of our single detection models. It can be seen that the larger the attack magnitude is, the better the defense results is. When the attack magnitude is 0, where there exists no attacker, all defense models result in random guess accuracy.

Furthermore, when the attack magnitude is small, some defense models do not perform as well. This is innocuous in the distributed learning scenarios, as when the attack magnitudes are too small to fool the detection models, they do not achieve visible attacking impact on the overall distributed learning system either. And if an uploaded vector is only a little different from others, we can't tell if it really is an attack vector, or it is from an honest collaborators whose dataset is different.

2) Multiple Attacker Defense: Figure 7 shows the performance of our defense models on multiple attackers. The results are achieved under the scenario where there are 10 peers, and 2 of them are attackers. We test 500 runs on each defense model, and this figure demonstrates the number of times 0, 1, or 2 attackers among all attackers are caught, respectively. Comparing with Figure 6, the accuracy of the defense models decrease when there are multiple attackers. This is intuitive, as more than one attackers leverage the entire attack vector for single attacker, the uploaded vector would differ less from the honest peers'.

D. Heritage Factor

As the definition states, the Heritage Factor is the metric that characterizes the percentage of the "legacy", Figure 6d)

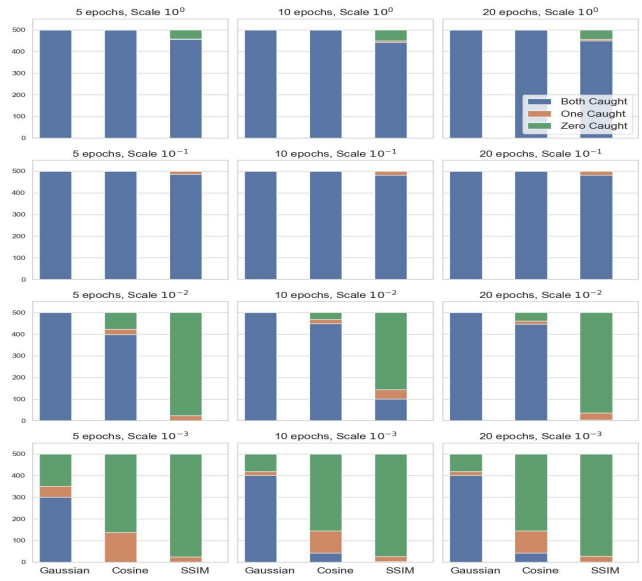

Fig. 7. Performance result of $N = 10$, 2 attackers The blue, orange, and green portions of the same bar indicates the number of times 2, 1, and 0 attackers are caught when there are in total 500 number of detection runs.

demonstrates the correlation between the Heritage Factor and the classification accuracy of honest peers. It suggests that the performance of the honest peers are in the positive relation with the Heritage Factor. Some other interesting experimental scenarios, including different number of total peers, different number of attackers..,etc, are also conducted, but due to the space limitation, those are not shown in this paper.

VII. Conclusion

In this paper, we study the attack-model-agnostic countermeasures in distributed learning system based on the fact that it's unrealistic to gain knowledge on the attack models that would be launched prior to the deployment of the distributed learning system. We explore existing threat and detect models, and propose the Drop-one and Structural Similarity (SSIM) defenses that analyze both the current and historic gradients to depict the attackers for poisoning attacks in distributed learning. A new concept, the Heritage Factor is presented that characterizes the false parameter propagation among the entire distributed learning system. Such definition also enables us to measure how "helpful" that one collaborator is to other collaborators and to the system. We verify through extensive experiments the effectiveness of the proposed attack-model-agnostic countermeasures over a wide range of attack magnitudes. To the best of our knowledge, our work is the first to systematically study the attack-model-agnostic countermeasures in distributed machine learning system.

Acknowledgement

This work is supported in part by the United States National Science Foundation (NSF) under grants CNS-2006998 and CNS-1837034. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF.

References

[1] K. Chang, N. Balachandar, C. Lam, D. Yi, J. Brown, A. Beers, B. Rosen, D. L. Rubin, and J. Kalpathy-Cramer, "Distributed deep learning networks among institutions for medical imaging," *Journal of the American Medical Informatics Association*, vol. 25, pp. 945–954, Aug. 2018.

[2] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, S. Zeitouni, F. Koushanfar, A.-R. Sadeghi, and T. Schneider, "FLAME: Taming backdoors in federated learning," in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), pp. 1415–1432, USENIX Association, Aug. 2022.

[3] C. Wu, X. Yang, S. Zhu, and P. Mitra, "Mitigating backdoor attacks in federated learning," *arXiv preprint arXiv:2011.01767*, 2020.

[4] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Byzantine-tolerant machine learning," *arXiv preprint arXiv:1703.02757*, 2017.

[5] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 27–38, 2017.

[6] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948, PMLR, 2020.

[7] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, pp. 634–643, PMLR, 2019.

[8] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1605–1622, 2020.

[9] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] C. Xie, O. Koyejo, and I. Gupta, "Generalized byzantine-tolerant sgd," *arXiv preprint arXiv:1802.10116*, 2018.

[11] S. Mahloujifar, M. Mahmoody, and A. Mohammed, "Universal multi-party poisoning attacks," in *International Conference on Machine Learning*, pp. 4274–4283, PMLR, 2019.

[12] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, pp. 5650–5659, PMLR, 2018.

[13] M. El El Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," *arXiv e-prints*, pp. arXiv–1802, 2018.

[14] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *Big Learning NIPS Workshop*, vol. 6, p. 2, 2013.

[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[16] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321, ACM, 2015.

[17] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, pp. 693–701, 2011.

[18] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[19] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, p. 18, 2010.

[20] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, vol. 6, 2017.