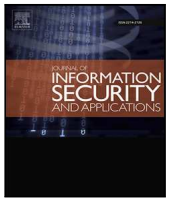
Journal of Information Security and Applications 82 (2024) 103739

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa



Attack-model-agnostic defense against model poisonings in distributed learning☆

Hairuo Xu, Tao Shu *

Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, 36849, United States

ARTICLE INFO

Keywords:
Distributed/federated learning
Attack-model-agnostic defense
Heritage factor
Threat and detection models

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.

1. Introduction

In recent years we have witnessed the initial success of machine-learning-based Artificial Intelligence (AI) in many application domains, such as image processing [1,2], natural language processing [3,4], autonomous driving [5,6], gaming [7–9], medical science [10–12] and public safety [13].

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 [14]. 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 parallelized 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.

☆ A preliminary version of this paper has been presented at *IEEE International Conference on Ubiquitous Intelligence and Computing (UIC) 2022* (Xu and Shu, 2022).

* Corresponding author.
E-mail address: tshu@auburn.edu (T. Shu).

https://doi.org/10.1016/j.jisa.2024.103739

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, when facing attack models different from the one that they were designed against, these countermeasures achieve limited defense effectiveness. For example, the defense methods described in [15,16] are aiming to defense the targeted attack models that insert a backdoor to mis-classify a certain label. However, when facing an untargeted attack that 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 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, which 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 untargeted attack countermeasures such as Krum [17] and AFA [18], and targeted attack countermeasures such label flipping attack and backdoor attack defenses. 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 and targeted attack models, and compare their effectiveness under different attack settings. A preliminary version of this paper has been accepted by the *IEEE International Conference on Ubiquitous Intelligence and Computing (UIC) 2022* [19]. 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 both the MNIST and CIFAR-10 datasets 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. In addition to the UIC version, we also evaluate the performance of the attack-model-agnostic defense methods, including the Krum, AFA, DropOne, SSIM, PCA-cluster defense, and an attack-specific defense, the CRFL defense method in distributed learning scenarios under both untargeted attack including the random noise attack, gradient ascent attack, and the targeted attacks including the label flipping attack and the backdoor attacks. Our extensive experiments confirms our observation that the attack-model-agnostic defense methods can indeed serve as a first line of defense that alleviate the zero-day vulnerability of the system.

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.

2. Related works

2.1. 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 [20,21], and the untargeted attack, whose goal is to prevent the distributed learning system from converging to its optimal solution as described in [22–25]. 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.

2.2. Defense methods

Based on the potential attack models, the defense method could also be divided into two categories. The targeted defense proposed in [15,16] were based on the prior knowledge of the attack models. And more generally, Krum and Multi-Krum [17], AFA [18], Trimmed-Mean [24], Median [26] and Bulyan [27] are proposed to counter the untargeted attacks in distributed learning systems.

These untargeted defense methods could be categorized into two groups. The first group include Krum, Multi-Krum and AFA, which consider each of the gradient (local update) as a whole part and try to recognize the outlying ones. Particularly, Krum regards the gradients that are closest to its $n - m - 2$ neighboring gradients in the Euclidean norm space as honest (where n, m are the total number of workers, and the upper bound of the adversaries, respectively), and only allow these gradients to be aggregated by the server. Multi-Krum utilizes Krum to select c gradients such that $n - c > 2\ m + 2$, and then takes the average of the selected c gradients to update the server. AFA computes the cosine-similarities among the gradients, and eliminate the ones with the out-of-bound cosine similarity scores.

The second group include Trimmed-Mean, Median and Bulyan, which consider the gradients from a dimension-wise perspective. For each dimension of gradients, Trimmed-Mean removes the β largest and smallest values, and takes the average of the rest to update that dimension. Median updates the server by the result of computing the median of the gradients in each dimension. And Bulyan can be seen as a mixture of Multi-Krum and Trimmed-Mean, as it first selects θ gradients as Multi-Krum, and then perform Trimmed-Mean on the selected set.

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.

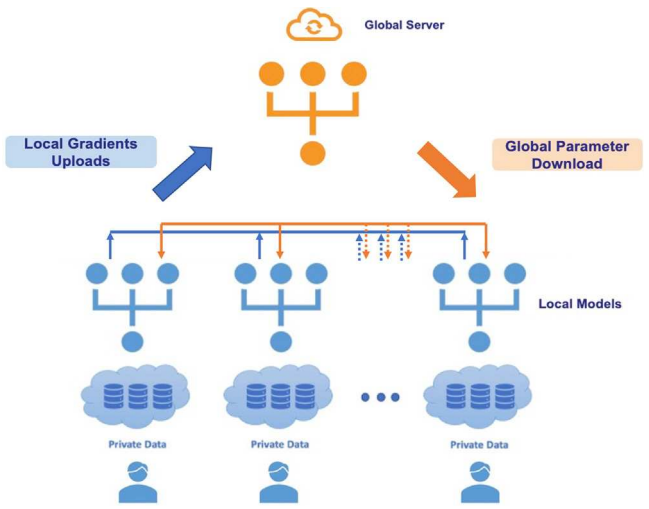


Fig. 1. High-level system architecture.

3. Model description and problem formulation

3.1. Overview

Fig. 1 illustrates the high-level architecture of our distributed learning system. It is an abstract example of the parameter server [28] system. The system contains one server, which is responsible for maintaining the global parameters, and N collaborators (also referred to as workers in the following text). There exists a 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.

3.2. Local training

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

Each worker i maintains a local set of its neural network parameters, denoted as $\mathbf{w}_k^i$. The worker starts training by downloading, and replacing all local parameters with the global parameters from the server. After that, the worker trains locally for one local epoch with Stochastic Gradient Descent optimization [29]. In this epoch, the worker should train on all the local data, instead of certain mini-batches. This single-epoch training produces a set of gradients, denoted as $\mathbf{g}_k^i$:

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

where $\mathbf{g}_k^i$ denotes worker i's true local gradients trained with SGD at epoch k, and $\mathbf{w}_k^i$ denotes the local weights (parameters) of worker i at epoch k after the local training. The worker will then upload the gradients $\mathbf{g}_k^i$ to the parameter server, and wait until the next iteration.

All the other workers will perform the same process of downloading and replacing the local parameters by the global parameters from the server, train locally for one epoch, stop and upload the gradients to the server. It is worth noting that such procedure is done simultaneously, as each worker has its own local model which is capable of training independently. The iteration continues until all the workers have trained locally for one epoch and uploaded their gradients to the server. The server will update the global parameters with the gradients received according to an aggregation rule, and then the next iteration starts, where each worker will download the updated global server and train locally for the second epoch.

3.3. Parameter server

The parameter server maintains the global parameters, denoted by $\mathbf{w}_k^{\text{global}}$. In each iteration, the server receives a set of gradient vectors, one from each worker, and updates the global parameters by the following FedAvg [30] aggregation equation:

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

where η denotes the learning rate, $k-1$ and k denote the iteration number, and $\mathbf{g}_k^i$ denotes the local gradients upload by worker i at epoch k. As shown in Fig. 1, the parameter update on the server completes the current iteration, and the next iteration starts.

3.4. Model-poisoning attack models

As will be clear shortly in Section 4, we explore a full array of the attack models, including the untargeted attacks such as random noise attack, gradient ascent attack; and the targeted attacks such as label flipping attack and backdoor attacks. While different attack models obtain different attacking strategies, in general, the essence of all these attack methods could be summarized as injecting an attack vector into/replacing the real local updates (or local training gradients) that is uploaded to the global server for aggregation.

In light of the above commonality shared by all model-poisoning attack methods, we formulate the general form of attack mechanisms by adding an attack vector into the local gradients, i.e.,

$$\widetilde{\mathbf{g}}_k^i = \mathbf{g}_k^i + \boldsymbol{\epsilon} \tag{3}$$

where $\widetilde{\mathbf{g}}_k^i$ denotes the poisoned local update to be uploaded to the server, $\boldsymbol{\epsilon}$ denotes the attack vector, and $\mathbf{g}_k^i$ denotes the true gradients. However, note that in many cases the attacker could just upload $\boldsymbol{\epsilon}$, whereby $\mathbf{g}_k^i$ is simply set to be a zero vector.

Under the above general form of attack models, we propose a new metric, the *heritage factor*, to characterize how much a reported local update deviates from its true local training outcome. In particular, we rewrite Eq. (3) as follows:

$$\widetilde{\mathbf{g}}_k^i = \rho \cdot \mathbf{g}_k^i + (1-\rho) \cdot \boldsymbol{\epsilon} \tag{4}$$

where we define $0 \le \rho \le 1$ to be the heritage factor, which reflects the following insight on model-poisoning attacks: If there exists no attacker, the distributed learning system could help each of the collaborators (workers) to learn a more generalized model and converge faster [31,32]. Intuitively, such collaborative learning procedure could be seen as an inheritance. The prior workers (or the workers from previous iterations) train their local models, and upload the gradients to the server, where such gradients could be regarded as the "legacies". These "legacies" are then aggregated by the parameter server, and "inherited" by the later workers (or the workers in later iterations) during the global parameters download as "heritage". In this paper, we assume that all honest workers are uploading their "legacies" 100%, but an attack would only upload a part of its "legacy", supplemented as another part by an attack vector $\boldsymbol{\epsilon}$. The heritage factor ρ just characterizes how much "legacy" is contained in a local update, or equivalently, the percentage of the true gradients that will be uploaded and inherited.

When ρ is 1, no attack vector is injected 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

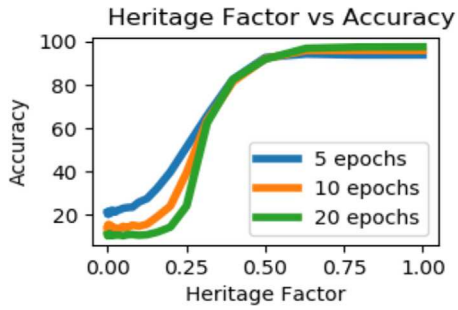


Fig. 2. Heritage factor evaluation: the correlation between the heritage factor and the performance of the overall distributed learning system.

be the attack vector. We evaluate the impact of the heritage factor to the overall distributed learning system shown in Fig. 2, and the experimental results validate our analysis above.

In particular, from the figure it can be observed that the performance of the honest workers has a positive correlation with the heritage factor. On one hand, if the heritage factor is 1, i.e., all workers are sharing their local updates without any mask, then the overall system is converging very quickly to the optimal solution. However, sharing the local updates directly could be risky, as it could lead to the leakage of their local training dataset, and therefore causing a privacy problem. On the other extreme, having the heritage factor of 0 would prevent the entire distributed learning system from converging, as all the local updates are completely hidden, which violates the original intention of distributed learning. In practice, we aim to choose the proper heritage factor that will not only protect the worker's privacy, but also enforce the original purpose of the collaboration of the distributed learning.

3.5. Problem definition

In this paper, we aim to develop attack-model-agnostic defense methods that are capable of filtering out poisoned local updates at the server, so that these updates will not participate in the model aggregation and hence are prevented from infecting/poisoning subsequent local model training. In general, the proposed defense methods could be seen as a black-box function that takes all local updates in one iteration as input, and produces a binary vector where each element denotes whether the corresponding local update would be included in the global server's aggregation. In our future work, the output of such a function could be improved to include a confidence score (ranging between 0 and 1) for each local update that indicates the weight of the update in the server's aggregation, so that a "soft" filtering technique, where the global server will take the weighted sum of all local updates multiplying their corresponding weights (or confidence score), instead of the "hard" filtering provided by our current binary vector output, in which the global server eliminates certain local updates completely, becomes feasible. However, such a "soft" filtering is out of the scope of this paper, and here we will only focus on the binary vector output and hard filtering.

In particular, an attack-model-agnostic defense method can be symbolically represented as the following function,

$$f([\mathbf{g}_k^1, \mathbf{g}_k^2, \dots, \mathbf{g}_k^N]) = \mathbf{b}. \tag{5}$$

where f denotes the filtering function, $[\mathbf{g}_k^1, \mathbf{g}_k^2, \dots, \mathbf{g}_k^N]$ denotes the list of uploaded local updates from all workers in iteration k, and the output $\mathbf{b}$ is a binary vector in which each element indicates whether the corresponding local update is accepted or rejected in the global aggregation. As will be clear shortly in Section 4, by exploiting the commonality of model-poisoning attacks along different directions, we will propose a variety of realizations for function f (i.e., a variety of attack-model-agnostic defense mechanisms) in Section 5.

4. Commonality presented by model-poisoning attacks

Our goal is to develop attack-model-agnostic defense mechanisms that are able to counter a wide spectrum of model-poisoning attacks. To this end, such a defense mechanism must target/exploit certain common features that are presented by all these model-poisoning attacks. Following this logic, in this section we make observation on the behavior of a wide range of representative model-poisoning attacks, with the intention to identify the common features presented by them when their attacks are successful. These observed common features will lead to our subsequent development of attack-model-agnostic defense mechanisms, as elaborated in Section 5.

4.1. Single attacker

Similar to the honest workers, 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 is 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 workers 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 categories of model poisoning attack, the untargeted attack, including the random noise attack and the gradient ascent attack; and the targeted attack, including the label flipping attack and the backdoor attack. We elaborate their strategies in fabricating the poisoning vectors below.

4.1.1. Random Noise Attack (RNA)

The random noise attack fabricates the poisoning vector by randomly generating a vector according to Normal distribution. With its own local dataset, the attacker approximates the μ and σ^2 of the local gradients from its first round of local training. After that, following the Normal distribution, the attacker randomly generates the attack vector and upload it to the global server as

$$\widetilde{\mathbf{g}}_k^i = \boldsymbol{\epsilon} \sim \mathcal{N}(\phi \cdot \mu, \sigma^2). \tag{6}$$

where $\boldsymbol{\epsilon}$ is uploaded to the parameter server as the true gradients, and ϕ denotes the attack magnitude. In experiments, we observe the impact of attack magnitude on the distributed learning system by changing ϕ.

4.1.2. Gradient Ascent Attack (GAA)

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** (negative gradient descent) optimization. The resulting gradients, serving as the poisoning vector, will be uploaded to the server. Similar to Eq. (1), the gradients is computed by subtracting the parameters downloaded from the parameters after gradient ascent training in current epoch, i.e.,

$$\widetilde{\mathbf{g}}_k^i = \boldsymbol{\epsilon} = \mathbf{w}_k^i - \mathbf{w}_{k-1}^i, \tag{7}$$

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

It is worth noting that the fabricated ascending gradients is calculated not w.r.t. the attacker's local parameters but the global parameters from the previous epoch. Such strategy would enhance the attack strength since the global parameters is an aggregation of local updates from all workers, and the ascending gradients calculated from those would have the higher potential of impacting all workers. In contrast,

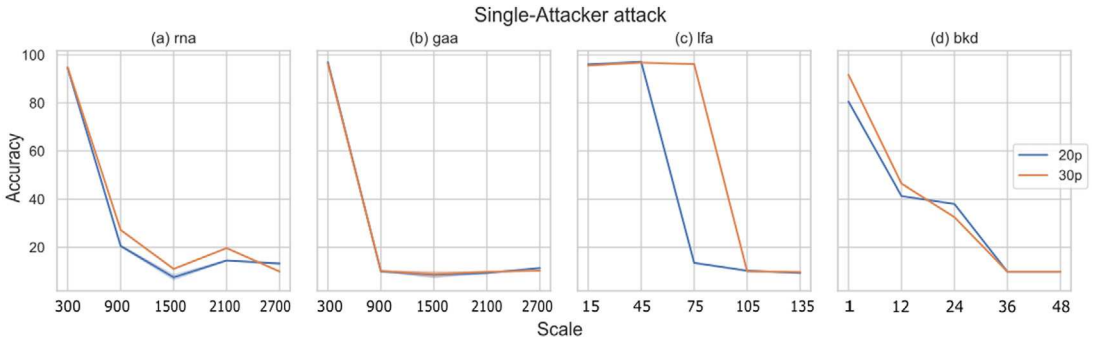


Fig. 3. Performance results of $N = 20$ (blue) and $N = 30$ (orange), under (a) random noise attack, (b) gradient ascent attack, (c) label flipping attack and (d) backdoor attack. The x-axis denotes the magnitude/scale calculated as the multiples of the true gradients. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the ascending gradients calculated based on the attacker's own parameters would lead to less impact on other workers due to the potential difference with other worker's parameters and data distribution.

In order to compare the gradient ascent attack with the random noise attack method, in some of our experiments, we multiply the fabricated attack vector calculated with the corresponding attack magnitude in the random noise attack. With this subtle change in experiments, the gradient ascent attack can be considered as a special case of the random noise attack, in which the injected random attack vector is the carefully computed to include the ascending gradients.

4.1.3. Label Flipping Attack (LFA)

The label flipping attack is one of the targeted attack models whose goal is to degrade the classification performance of data examples from a certain class, and such class is referred to as the attack target. In our case, since we use the classification task to demonstrate the attack and defense effectiveness, the label flipping attacker's goal is to destroy the recognition of a specific class. The attacker first pick the target class, say class 7 of the MNIST dataset, and change part (or all) of their labels to 1. After that, the attacker trains its local model with the polluted local dataset, and uploads the resulting gradients to the global server. Notice that although the primary goal is to destroy the recognition of the target class, it is possible that it could also impact the performance of other classes, or even the overall tasks as well.

4.1.4. Backdoor attack (BKD)

The backdoor attack also belongs to the targeted attack category. In this paper, we take the strategy from backdoor paper [20], and add a little modification. This paper explains the usage of semantic features, for example, the *green* car, of the car *with strips*, and their attack was focused on the data with such specific features. However, since we are using the MNIST and CIFAR-10 dataset for simulation, it is difficult to manually specify the features of the data, and therefore, in this paper, our backdoor attack method is tweaked as following. The attacker first add a patch to the target local data at the bottom right corner, and then change the label of the target data to another. The attacker will then upload the local gradients trained on the polluted dataset to the parameter server. Similar to the label flipping attack described earlier, the backdoor attack also focuses on a specific class, and may or may not affect the overall distributed learning system.

4.2. Colluded 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.

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 fixed overall magnitude of the attack vectors, the group of attackers will collaborate to produce multiple attack vectors, pretending to be the true gradients. Denote the magnitude of the overall attack vector as ϕ, and denote the number of attackers as N_a, The attack vector that each attacker will generate is:

$$\tilde{\mathbf{g}}_k^i = \epsilon \sim \mathcal{N}(\frac{\phi}{N_a} \cdot \mu, \sigma^2). \tag{8}$$

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

In Colluded Gradient Ascent Attack, each attacker computes the global **ascending** gradients, and upload (part of) the ascending gradients to the server. In this case, it is expected that the final aggregated ascending vector would be more generalized and has a higher attack success rate as it is now not computed solely on one worker's local data.

Similarly, in the Colluded Label Flipping Attack and the Backdoor Attack, the attackers had agreed in advance on the source and target. They will then perform the polluted label injection or backdoor patch injection, local training, and finally upload the falsely trained gradients to the server.

4.3. Evaluation and observation of the attacks

4.3.1. Single-attacker case

Experiments are conducted simulating the scenario where among all the collaborators, only one of them is the attacker. We simulate such scenario with four different attack methods: two untargeted: random noise attack, gradient ascent attack, and two targeted: label flipping attack and backdoor attack. Fig. 3 demonstrates the correlation between the magnitude (scale) of the attacking vector versus the overall distributed learning system.

For the two untargeted attack methods demonstrated in Fig. 3(a) and (b), an observation could be made that the overall system accuracy is inversely proportional with the scale of the injected attack. When the injected attack is large enough, the overall system could not converge. Furthermore, by comparing these two sub-graphs, it is suggested that under the same attack magnitude, the gradient ascent attack results in a stronger impact on the distributed learning system. Such phenomena is quite intuitive, as the gradient ascent attack is attacking towards the negative training (or negative gradient descent) direction while the random noise attack could be seen as attacking towards the random directions.

In contrast, the two targeted attacks shown in Fig. 3(c) and (d) obtain the same commonality that the larger the magnitude is, the

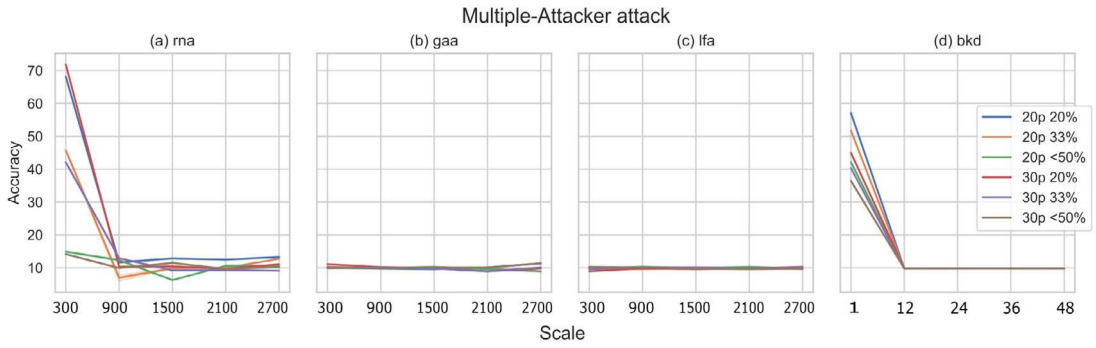


Fig. 4. Performance results of $N = 20$ and $N = 30$, in which different percentages (20%, 33% and <50%) of them are attackers, respectively.

worse accuracy it achieves. However, in comparison with the untargeted attacks, these two targeted attacks requires less attack magnitude to penetrate the overall distributed learning system due to the fact that these two algorithms are well-designed to attack a certain target. For the untargeted attacks, in our case, it requires the magnitude of >900 in order to diverge the system, while having the magnitude of only >75, and >30 could fail the learning in the label flipping attack, and backdoor attack, respectively.

4.3.2. Multiple-attacker case

Fig. 4 demonstrates the simulation results when there are 20 and 30 collaborators and among which 20%, 33% and <50% of them are attackers, respectively. In the multiple-attackers scenario, the attackers share the overall attack magnitude. An observation could be made that the more attackers there are among all workers, the worse the overall performance of the distributed learning is. Such observation is common for all four attack methods, although sub-graphs (b) and (c) does not seem to follow, this is due to the fact that the attack magnitude in this figure is too large for them to handle. Same experiments on a smaller magnitude could be found in the Appendix where such pattern does appear.

After the evaluation and analysis of the full array of attack models elaborated above, we make the observation that a sufficiently large attack magnitude (typically at least ten times of that of the true update, as evidenced in our experiment results) is the common and dominant factor in deciding whether an attack could successfully penetrate the distributed learning system, regardless of the exact attack model. And therefore, in the following section, we propose the attack-agnostic defense methods to counter such a dominant factor.

5. Attack-agnostic defense methods

In this section, attack-model-agnostic defense mechanisms are developed by exploiting the observed fact that the magnitude of a model-poisoning attack vector, regardless of the specific attack models, has to be many times greater than that of a regular local update in order to make the attack successful. Such a big magnitude will make the poisoned update an outlier among all local updates reported to the server. As such, certain statistical metric may be defined and then used to filter out these outliers, with the intention to exclude those poisoned updates from the server's model aggregation. In the following, several such statistical metrics are proposed and used for this purpose, leading to several attack-model-agnostic defense methods.

In particular, we first propose two novel defense methods: the DropOne method, and the SSIM (Structural Similarity) method. We then look into two existing defense methods, Krum and AFA (cosine similarity), which were originally proposed to counter the untargeted attacks. We analyze these two algorithms, and explore their suitability to counter targeted attacks, so as to adapt them to attack-model-agnostic defenses (i.e., applicable to both targeted and untargeted attacks). Finally, we also present a PCA-based defense algorithm that was originally designed to counter the label flipping attack (a targeted attack model). We argue that such a PCA-based method may also be adapted for attack-model-agnostic defense. The performance of all proposed defenses are subsequently evaluated in Section 6 under the full set of attack models defined in Section 3.

5.1. Drop One Detection

As the name suggest, the Drop One Detection method is applicable when there is only one attacker, but in reality, such defense method could be applied multiple times to achieve the desired defense results. The Drop One Defense starts with putting 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 worker 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. The intuition is that the large magnitude, which is the dominant factor of attacks methods, would naturally increase the group's variance, and therefore, the group of local updates with the smallest standard deviation (or square root of the variance) can be thought of as the group which has the attack vector dropped.

5.2. Series Structural Similarity

The Structural Similarity (SSIM) was first proposed in [33] 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 common underlying structures that lead to global convergence. 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 worker i by $\mathbf{T}^i = \{\mathbf{g}_1^i, \mathbf{g}_2^i, \dots, \mathbf{g}_k^i\}$, where the $1, 2, \dots, k$ represents the epochs number. The SSIM defense method computes the structural similarity score [33] to analyzes the trace of each worker's local updates and find out the suspicious workers. The intuition is that the SSIM score calculated between the reference and the attack vector (with large magnitudes) would be smaller, in contrast, an honest worker will have a larger SSIM score with the reference.

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.

5.3. Gaussian Detection (Krum)

The Krum detection method is proposed in [17] 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 Krum defense starts by calculating the gradients mean μ_k and the standard deviation σ_k.. Then, for each uploaded vector, its distances to the mean is calculated, and is represented by the multiples of standard deviation σ_k as

$$\theta_k^i = (\mathbf{g}_k^i - \mu_k)/\sigma_k \tag{9}$$

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 threshold used is observed from the experiments to obtain the best accuracy.

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. Such focus on the magnitude is directly aiming on the commonality of the attack methods observed in the previous section, and therefore, achieves the best and most robust defense results as shown in Section 6.

5.4. Cosine similarity detection (AFA)

AFA [18] 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 workers, 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

$$cos(\beta) = \frac{\mathbf{g}_k^i \cdot \mu}{\|\mathbf{g}_k^i\| \cdot \|\mu\|} = \frac{\sum_{j=1}^n \mathbf{g}_{k_j}^i \cdot \mu_j}{\sqrt{\sum_{j=1}^n \mathbf{g}_{k_j}^{i2}} \cdot \sqrt{\sum_{j=1}^n \mu_j^2}}. \tag{10}$$

Originally, a smaller cosine similarity score indicates the less similarity from the mean, and the corresponding collaborator who obtain a small score is more suspicious. However, when there exists a large-magnitude attack vector, it would modify the original mean vector and drag it closer to the attack vector itself. Even with such mean-dragging issue, we can still filter out the attack vector based on the fact that the cosine similarity score between the mean and attack vector will often differ from those calculated between the mean and the honest worker's local updates.

5.5. PCA-based label flipping attack defense [34]

The PCA-based label flipping attack defense algorithm aims to counter the attackers in Distributed/Federated learning by first collecting all gradients for the common Neural Networks' output layer on a specific class label (say label 1). After that, PCA is performed on all gradients for dimension reduction, and we cluster the results into two groups, where one of them is regarded as the attacker group, and their local updates are ignored. Such procedure is executed for each possible output label, and thus completes the defense for label flipping attack.

Although this defense method was originally proposed to counter against the label flipping attack, we argue that such model actually falls into the category of attack-model-agnostic defense methods, as it is focusing on the structure of the training model from the perspective of learning task, instead of a certain attack model. Particularly, PCA is known to extract the underlying features in a group, and in which the magnitude of the local updates could also be considered as one of the principle components. Therefore, the clustering after the PCA would separate the local updates with different magnitudes (and other features) into different groups, and hence achieve our defensive goal. We will show that such PCA-based defense algorithm can also counter against other attacks in Section 6 to prove our analysis that this really is an attack-agnostic defense model.

6. Evaluation

6.1. Dataset and setup

We conduct extensive experiments using both the MNIST [35] and CIFAR-10 [36] datasets to verify the effectiveness and generalizability of our proposed attack-model-agnostic defense mechanisms. The MNIST dataset contains 60,000 black and white hand-written digit images in the training set, and 10,000 in the testing set. The CIFAR-10 dataset consists of 50,000 3 channel RGB object images in the training set, and 10,000 in the testing set. We implement the distributed learning framework with one parameter server and multiple local workers, each maintains the same neural networks architecture. The training data are shuffled and separated randomly into each worker.

We use PyTorch [37], version 1.13.1, 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.

6.2. Defense evaluation on MNIST

In this section, we perform extensive experiments of our defense methods under different scenarios. We separate the defense results into three subsections, where each one explores a potential changing factor in reality.

6.2.1. Validation of the attack-model-agnostic property of the defenses

We have tested all aforementioned defense methods against a variety of attack models as defined in Section 3, including both untargeted and targeted attacks. It could be observed in Fig. 5 that all of the defense: Krum, Afa (cosine similarity), DropOne, Ssim, and the PCA-based methods are capable of defending against the random noise attack (rna), gradient ascent attack (gaa), label flipping attack (lfa), and backdoor attack (bkd). Such observation validates our analysis in Section 5 that these defense algorithms really are attack-model agnostic (i.e., their effectiveness do not rely on a specific attack model, but provide a wide-spectrum defense for most kinds of attack methods). The same observation can also be made under all other tested attack settings, as shown in Figs. 6–10.

6.2.2. Defense w.r.t. the attack magnitude

From Figs. 3–4, it can be observed that the magnitude of the injected attack vector is the dominant factor in deciding the ultimate attack

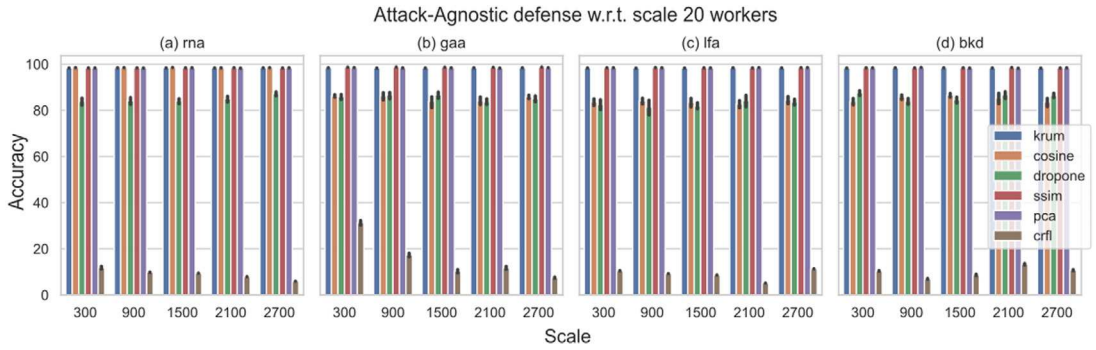


Fig. 5. Performance results of $N = 20$, single attacker, under four different attack methods (a) random noise attack, (b) gradient ascent attack, (c) label flipping attack, and (d) backdoor attack; with six different defense method applied: Krum, AFA (cosine), DropOne, SSIM (structural Similarity), PCA, and CRFL respectively.

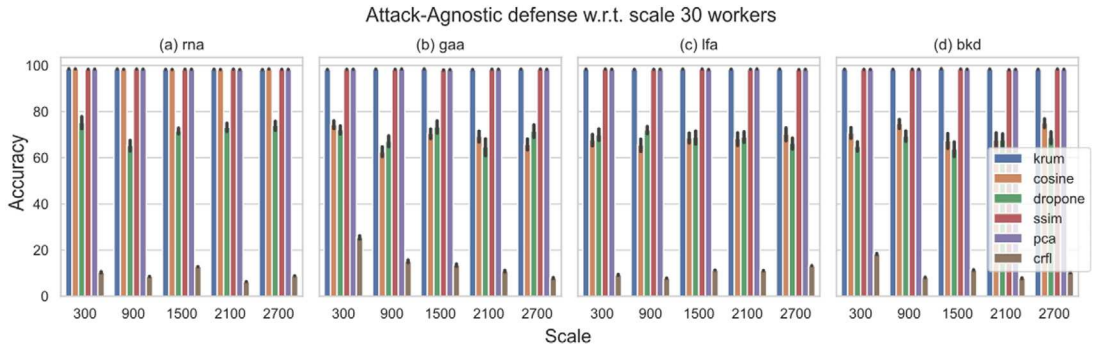


Fig. 6. Performance results of $N = 30$, single attacker, under four attacks and six defense methods, respectively.

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 on the four representative attack methods.

Figs. 5 and 6 illustrate the defense performance of the six defense methods. The x-axis represents the scale of the attack, and y-axis records the overall accuracy of the distributed learning system with single attacker after the corresponding defense mechanism is applied. Both figures suggest that the attack-model-agnostic defense methods: Krum, AFA, DropOne, Structural-Similarity, and PCA are able to defend against the tested four attack models. We also compare such attack-agnostic defenses with a representative attack-model-focused defense method, the CRFL (certified Robust FL against backdoor attack) model, and the result confirms our analysis earlier that the defense model proposed aiming to defend against one particular attack model often perform badly when facing other kind of attacks. Furthermore, the CRFL method is only able to defense against one backdoor attack model [20], when we test it with another backdoor attack method by tweaking [20] and adding a backdoor patch to the training samples described in Section 4.1.4, such defense stopped working. This observation again validates the existing limitation of attack-model-focused defense method, and in contrast proves the necessity of the attack-model-agnostic defense methods.

In this paper, although we are emphasizing the need of the attack-model-agnostic defense methods, each of such defense method also has their limitations. For example, both Figs. 5 and 6 suggest that the AFA (cosine similarity) and the Dropone method do not work as well as other defense methods such as Krum and Structural Similarity, and with the increment number of workers, such performance gap becomes more obvious.

6.2.3. Defense w.r.t. N (number of total workers)

In this subsection, we explore the correlation between the performance of the defense methods with respect to the total number of workers (N) under the fixed scenario of single attacker and attack magnitude. Figs. 7 and 8 illustrate our test results. The x-axis represents the total number of workers N, and the y-axis records the testing accuracy of the distributed learning algorithm. It is observed that other than the CRFL defense method (non-attack-agnostic) which does not work in any of the scenario, all the other attack-model-agnostic defense methods are able to defend against the four attacks. The best attack-model-agnostic defense methods in this scenario are Krum, SSIM, and PCA, as they are able to protect the distributed learning system and enforce it to converge to the original, attacker-free scenario's accuracy.

For the AFA and DropOne detection methods, it can be observed that, under a fixed number of attackers (in this case, there is only one attacker), the distributed learning system with more collaborative workers suffers more from the attack, no matter what the exact attack model is. Such observation may be against intuition at first, as the common thinking is that the more honest workers there are, the more correct references there are, and hence it is easier to differentiate the attackers. However, both the AFA and DropOne defense methods rely heavily on the statistics of the uploaded gradients, and therefore, if one attacker obtains a very large distance from the honest ones, it is highly possible that the attacker-uploaded gradients could drag the overall mean and standard deviation towards its direction, and consequently, forcing the defense methods to eliminate the honest workers instead. With more number of workers, it is highly likely that more honest workers are eliminated at each round of aggregation, leading to the observation described above.

6.2.4. Defense w.r.t. multiple colluding attackers

In this subsection, we evaluate the performance of the attack-model-agnostic defense methods under the scenarios of multiple attackers. As Fig. 4 suggests that while there exists multiple colluding attackers, the distributed learning system suffers more, therefore, obtaining the capability of defending against the multiple colluding attack is beyond important.

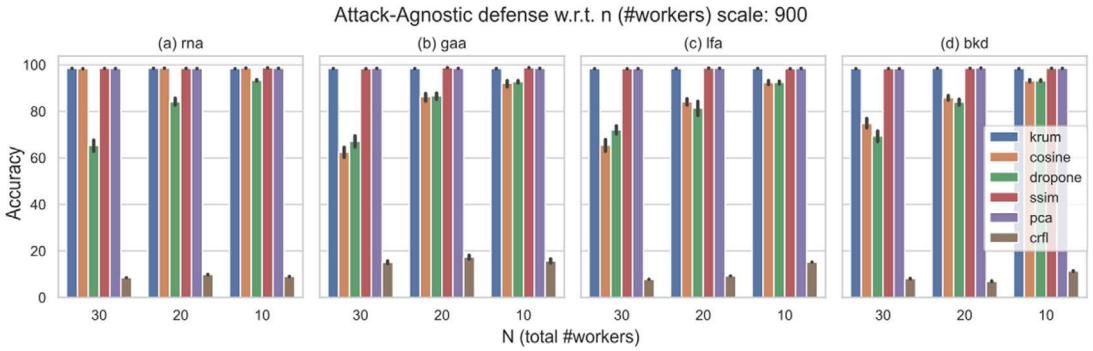


Fig. 7. Performance results of $N = 20$, single attacker with attack scale 900 with four attacks and six defense methods, respectively.

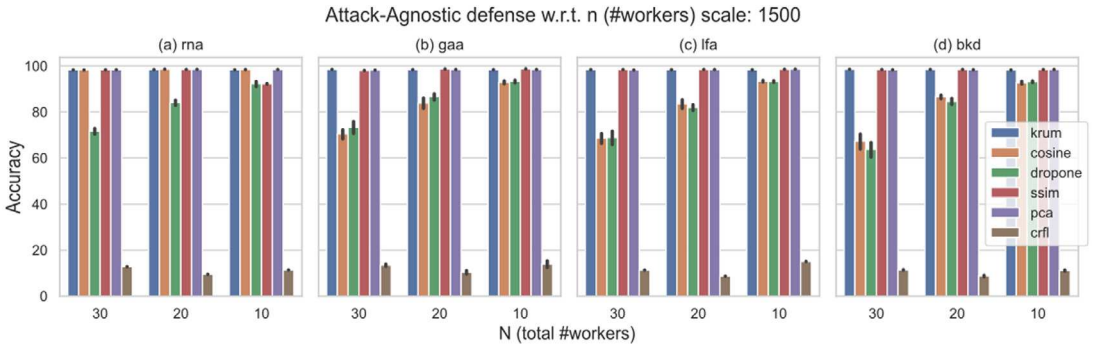


Fig. 8. Performance results of $N = 20$, single attacker with attack scale 1500 with four attacks and six defense methods, respectively.

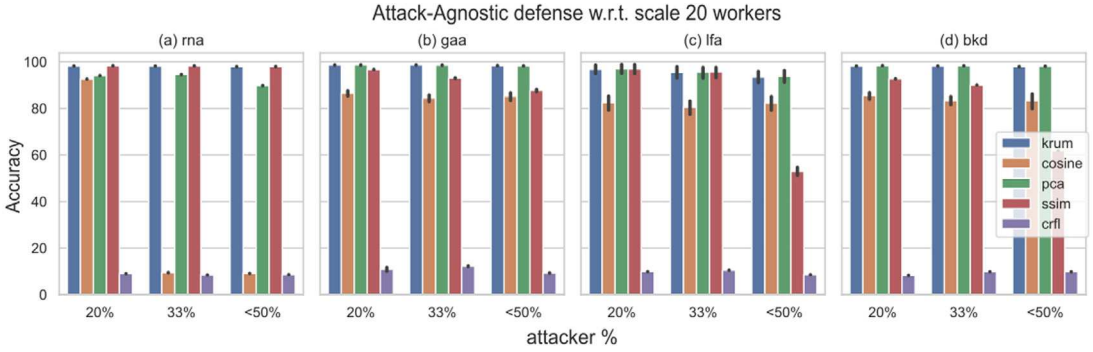


Fig. 9. Performance results of $N = 20$, different percentage of attackers with set attack magnitude with four attacks and six defense methods respectively.

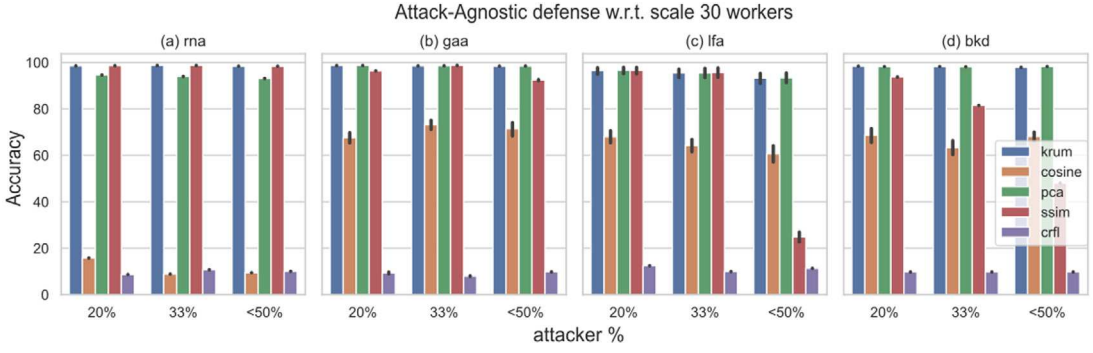


Fig. 10. Performance results of $N = 30$, different percentage of attackers with set attack magnitude with four attacks and six defense methods respectively.

The essential setup of this experiments is the same as the one from Fig. 4, where among N workers, 20%, 33%, and <50% of them are attackers. And the experiments are conducted with a set scale of 900. Figs. 9 and 10 illustrates our evaluation results where the x-axis represents the percentage of number of attackers to the total number of workers (i.e., m/N), and the y-axis records the performance of the overall distributed learning system. It can be seen that most attack-model-agnostic defense methods are still able to perform the defending

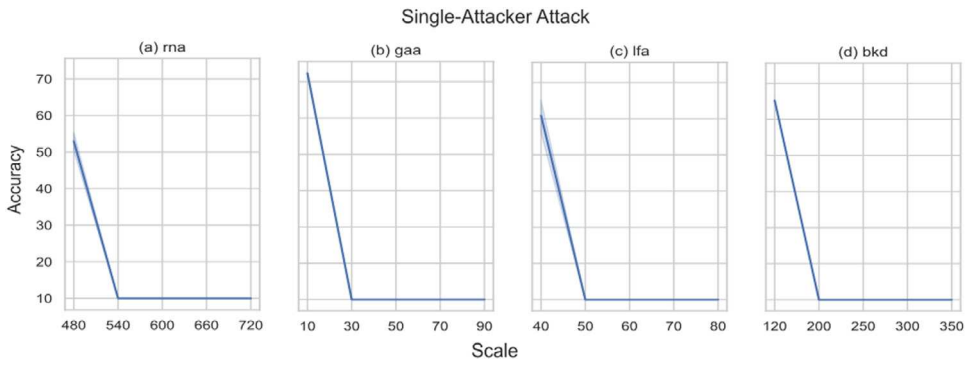


Fig. 11. Performance results of $N = 20$ workers, under (a) random noise attack, (b) gradient ascent attack, (c) label flipping attack and (d) backdoor attack on CIFAR-10 dataset. The x-axis denotes the magnitude/scale calculated as the multiples of the true gradients.

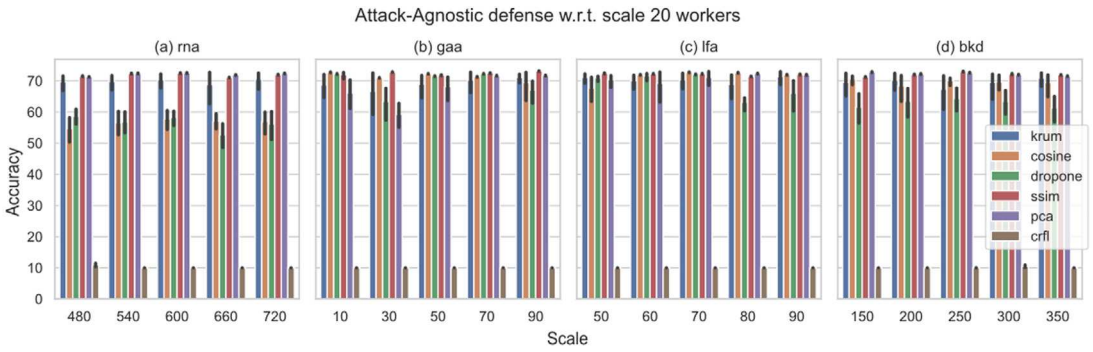


Fig. 12. Performance results of $N = 20$, single attacker, under four different attack methods (a) random noise attack, (b) gradient ascent attack, (c) label flipping attack, and (d) backdoor attack; with six different defense method applied: Krum, AFA (cosine), DropOne, SSIM (structural Similarity), PCA, and CRFL respectively on CIFAR-10 dataset.

functionality, however, we see that the AFA (cosine similarity) has the worst performance among all scenarios. Particularly, AFA failed to defend against the random noise attack in most cases. This could be due to the fact that random noise attack aims to attack at random directions, and having multiple random-directional gradients largely confuses the AFA defense which is mostly based on the angular distance between one gradient and the mean gradients vector.

Another observation is that when the number of attackers (m) is large, the SSIM defense method does not work as well either. This is because of the fact that SSIM randomly selects a certain percentage of workers' gradients as "reference", and then eliminate the workers with the worst SSIM score. When the attacker percentage is <50%, there is almost half attacker and half honest worker in the reference pool, making it difficult to differentiate the attackers from the honest workers.

6.3. Defense evaluation on CIFAR-10

In order to show that our proposed attack-model-agnostic defense mechanisms are also applicable on other dataset, in this subsection, we present our experimental results on CIFAR-10. In particular, we consider the scenario of defense w.r.t. the attack magnitude, which is the common and dominant factor of whether an attack could successfully penetrate the distributed learning system, as discussed at the end of Section 4.3.2. The other two experimental settings could be regarded as special cases of the selected defense w.r.t. attack magnitude scenario, where the scenario with the variable N (total number of workers) can be seen as introducing more honest workers to leverage the effect of an existing attack, and the scenario with the variable m (the number of colluding adversaries) can be seen as multiple colluding attackers sharing the total attack workload of a given attack magnitude.

The results of these experiments are shown in Figs. 11 and 12. It can be observed in Fig. 11 that when there is no defense, the distributed learning framework hardly survives any attack model, especially when the attack scale is large. The results in Fig. 12 verify again (as verified in our results based on MNIST dataset) that all the proposed attack-model-agnostic defense methods: Krum, Afa (cosine similarity), DropOne, Ssim, and the PCA-based methods are capable of defending against a variety of different attack models including the random noise attack (rna), gradient ascent attack (gaa), label flipping attack (lfa), and backdoor attack (bkd). These new experiments indicate that our proposed wide-spectrum defense mechanisms are also effective on different dataset other than MNIST, and hence verify the generalizability of these mechanisms.

7. Conclusion

In this paper, we study the attack-model-agnostic countermeasures in distributed learning system based on the fact that it is unrealistic to gain knowledge on the attack models that would be launched prior to the deployment of the distributed learning system. We introduced multiple existing threat and detection models, and propose the Dropone and Structural Similarity (SSIM) defense methods 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 between the child models in different iterations of distributed learning. 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 models. To the best of our knowledge, our work is the first to systematically study the attack-model-agnostic countermeasures in distributed machine learning system. Although the verification of the effectiveness of the proposed method is based on existing attack techniques (this is what it has to be), we expect that our proposed attack-model-agnostic defense mechanism

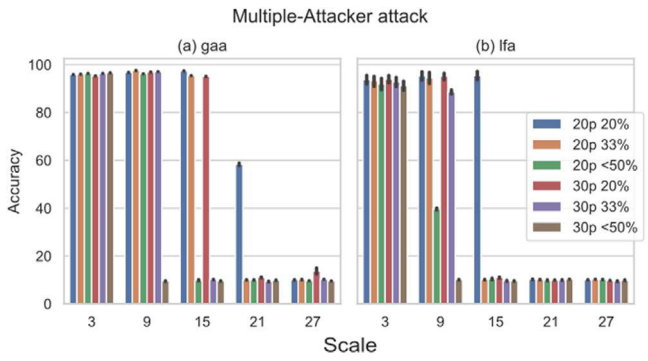


Fig. A.13. Evaluation of multiple attackers for (a) gradient ascent attack, and (b) label flipping attack.

is capable of defending against most of the future attack algorithms that take the approach of injecting a large-scale poisoning local updates into the server, just like the commonality shared by those existing attack models verified in our experiments.

CRediT authorship contribution statement

Hairuo Xu: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Tao Shu:** Funding acquisition, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Tao Shu reports financial support was provided by US National Science Foundation. Hairuo Xu reports financial support was provided by US National Science Foundation.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported in part by the United States National Science Foundation (NSF) under grants CNS-2308761 and CNS-2006998. 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.

Appendix. Multiple attackers evaluation with smaller magnitudes for GAA and LFA

This appendix aims to demonstrate the extended experiments we did for the LFA and GAA attacker under the multiple attackers scenarios, and Fig. A.13. We want to emphasize the pattern that more colluding attackers would definitely lead to worst performance of the distributed learning systems, which was not obvious enough in Fig. 4(b) and (c) because the attack magnitudes was too large so that even small number of attackers could sabotage the entire learning system.

References

[1] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Commun ACM 2017;60(6):84–90.
[2] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE; 2009, p. 248–55.
[3] Lagler K, Schindelegger M, Böhm J, Krásná H, Nilsson T. GPT2: Empirical slant delay model for radio space geodetic techniques. Geophys Res Lett 2013;40(6):1069–73.
[4] Floridi L, Chiriatti M. GPT-3: Its nature, scope, limits, and consequences. Minds Mach 2020;30(4):681–94.
[5] Niknam S, Dhillon HS, Reed JH. Federated learning for wireless communications: Motivation, opportunities, and challenges. IEEE Commun Mag 2020;58(6):46–51.
[6] Du Z, Wu C, Yoshinaga T, Yau K-LA, Ji Y, Li J. Federated learning for vehicular internet of things: Recent advances and open issues. IEEE Open J Comput Soc 2020;1:45–61.
[7] Vinyals O, Ewalds T, Bartunov S, Georgiev P, Vezhnevets AS, Yeo M, Makhzani A, Küttler H, Agapiou J, Schrittwieser J, et al. Starcraft ii: A new challenge for reinforcement learning. 2017, arXiv preprint arXiv:1708.04782.
[8] Lanctot M, Zambaldi V, Gruslys A, Lazaridou A, Tuyls K, Pérolat J, Silver D, Graepel T. A unified game-theoretic approach to multiagent reinforcement learning. Adv Neural Inf Process Syst 2017;30.
[9] Wang F-Y, Zhang JJ, Zheng X, Wang X, Yuan Y, Dai X, Zhang J, Yang L. Where does alphago go: From church-turing thesis to alphago thesis and beyond. IEEE/CAA J Autom Sin 2016;3(2):113–20.
[10] Wang Y, Bhattacharya T, Jiang Y, Qin X, Wang Y, Liu Y, Saykin AJ, Chen L. A novel deep learning method for predictive modeling of microbiome data. Brief Bioinform 2021;22(3):bbaa073.
[11] Wang Y, Jiang Y, Yao B, Huang K, Liu Y, Wang Y, Qin X, Saykin AJ, Chen L. WEVar: a novel statistical learning framework for predicting noncoding regulatory variants. Brief Bioinform 2021;22(6):bbab189.
[12] Wang Y, Zhao H, Sciabola S, Wang W. cMolGPT: A conditional generative pre-trained transformer for target-specific de novo molecular generation. Molecules 2023;28(11):4430.
[13] Anwar MZ, Kaleem Z, Jamalipour A. Machine learning inspired sound-based amateur drone detection for public safety applications. IEEE Trans Veh Technol 2019;68(3):2526–34. http://dx.doi.org/10.1109/TVT.2019.2893615.
[14] Chang K, Balachandar N, Lam C, Yi D, Brown J, Beers A, Rosen B, Rubin DL, Kalpathy-Cramer J. Distributed deep learning networks among institutions for medical imaging. J Am Med Inform Assoc 2018;25(8):945–54.
[15] Nguyen TD, Rieger P, Chen H, Yalame H, Möllering H, Fereidooni H, Marchal S, Miettinen M, Mirhoseini A, Zeitouni S, Koushanfar F, Sadeghi A-R, Schneider T. FLAME: Taming backdoors in federated learning. In: 31st USENIX security symposium. USeNIX security 22, Boston, MA; 2022, p. 1415–32.
[16] Wu C, Yang X, Zhu S, Mitra P. Mitigating backdoor attacks in federated learning. 2020, arXiv preprint arXiv:2011.01767.
[17] Blanchard P, Mhamdi EME, Guerraoui R, Stainer J. Byzantine-tolerant machine learning. 2017, arXiv preprint arXiv:1703.02757.
[18] Muñoz-González L, Biggio B, Demontis A, Paudice A, Wongrassamee V, Lupu EC, Roli F. Towards poisoning of deep learning algorithms with back-gradient optimization. In: Proceedings of the 10th ACM workshop on artificial intelligence and security. 2017, p. 27–38.
[19] Xu H, Shu T. Attack-model-agnostic defense against model poisonings in distributed learning. In: UIC 2022: 19th IEEE international conference on ubiquitous intelligence and computing, Haikou, Hainan, China, December 15–18. IEEE; 2022.
[20] Bagdasaryan E, Veit A, Hua Y, Estrin D, Shmatikov V. How to backdoor federated learning. In: International conference on artificial intelligence and statistics. PMLR; 2020, p. 2938–48.
[21] Bhagoji AN, Chakraborty S, Mittal P, Calo S. Analyzing federated learning through an adversarial lens. In: International conference on machine learning. PMLR; 2019, p. 634–43.
[22] Fang M, Cao X, Jia J, Gong N. Local model poisoning attacks to {Byzantine-robust} federated learning. In: 29th USENIX security symposium. USeNIX security 20, 2020, p. 1605–22.
[23] Baruch G, Baruch M, Goldberg Y. A little is enough: Circumventing defenses for distributed learning. Adv Neural Inf Process Syst 2019;32.
[24] Xie C, Koyejo O, Gupta I. Generalized byzantine-tolerant sgd. 2018, arXiv preprint arXiv:1802.10116.
[25] Mahloujifar S, Mahmoody M, Mohammed A. Universal multi-party poisoning attacks. In: International conference on machine learning. PMLR; 2019, p. 4274–83.
[26] Yin D, Chen Y, Kannan R, Bartlett P. Byzantine-robust distributed learning: Towards optimal statistical rates. In: International conference on machine learning. PMLR; 2018, p. 5650–9.
[27] El El Mhamdi M, Guerraoui R, Rouault S. The hidden vulnerability of distributed learning in byzantium. 2018, arXiv e-prints, arXiv–1802.
[28] Li M, Zhou L, Yang Z, Li A, Xia F, Andersen DG, Smola A. Parameter server for distributed machine learning. In: Big learning NIPS workshop. Vol. 6, 2013, p. 2.
[29] Bottou L. Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010. Springer; 2010, p. 177–86.

[30] McMahan B, Moore E, Ramage D, Hampson S, y Arcas BA. Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. PMLR; 2017, p. 1273–82.
[31] Shokri R, Shmatikov V. Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM; 2015, p. 1310–21.
[32] Recht B, Re C, Wright S, Niu F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems. 2011, p. 693–701.
[33] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP, et al. Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process 2004;13(4):600–12.
[34] Tolpegin V, Truex S, Gursoy ME, Liu L. Data poisoning attacks against federated learning systems. In: Computer security–ESORICs 2020: 25th European symposium on research in computer security, ESORICs 2020, Guildford, UK, September 14–18, 2020, proceedings, part i 25. Springer; 2020, p. 480–501.
[35] LeCun Y, Cortes C, Burges C. MNIST handwritten digit database. Vol. 2, AT&T Labs; 2010, p. 18, [Online]. Available: http://yann.lecun.com/exdb/mnist.
[36] Krizhevsky A, Hinton G, et al. Learning multiple layers of features from tiny images. 2009.
[37] Paszke A, Gross S, Chintala S, Chanan G. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. In: PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration. Vol. 6, 2017.