FL-Defender: Combating Targeted Attacks in Federated Learning

Najeeb Jebreel and Josep Domingo-Ferrer

Universitat Rovira i Virgili,
Department of Computer Engineering and Mathematics,
CYBERCAT-Center for Cybersecurity Research of Catalonia,
UNESCO Chair in Data Privacy,
Av. Països Catalans 26, 43007 Tarragona, Catalonia
{najeeb.jebreel, josep.domingo}@urv.cat

Abstract. Federated learning (FL) enables learning a global machine learning model from local data distributed among a set of participating workers. This makes it possible i) to train more accurate models due to learning from rich joint training data, and ii) to improve privacy by not sharing the workers' local private data with others. However, the distributed nature of FL makes it vulnerable to targeted poisoning attacks that negatively impact the integrity of the learned model while, unfortunately, being difficult to detect. Existing defenses against those attacks are limited by assumptions on the workers' data distribution, may degrade the global model performance on the main task and/or are ill-suited to high-dimensional models. In this paper, we analyze targeted attacks against FL and find that the neurons in the last layer of a deep learning (DL) model that are related to the attacks exhibit a different behavior from the unrelated neurons, making the last-layer gradients valuable features for attack detection. Accordingly, we propose *FL-Defender* as a method to combat FL targeted attacks. It consists of i) engineering more robust discriminative features by calculating the worker-wise angle similarity for the workers' last-layer gradients, ii) compressing the resulting similarity vectors using PCA to reduce redundant information, and iii) re-weighting the workers' updates based on their deviation from the centroid of the compressed similarity vectors. Experiments on three data sets with different DL model sizes and data distributions show the effectiveness of our method at defending against label-flipping and backdoor attacks. Compared to several state-of-the-art defenses, FL-Defender achieves the lowest attack success rates, maintains the performance of the global model on the main task and causes minimal computational overhead on the server.

Keywords: Federated learning · Security and robustness · Targeted poisoning attacks · Label-flipping attacks · Backdoor attacks

1 Introduction

Federated learning (FL) [18,24] enables building an ML model from private data distributed among a set of participating worker devices. In FL, the workers fine-tune a global model received from the server on their local data to compute local model updates that they upload to the server, which aggregates them to obtain an updated global model. This process is iterated until it converges to a high-quality global model. Therefore, FL improves accuracy, privacy and scalability, respectively, by learning from big joint data, keeping the workers' local data at their respective devices (*e.g.* smartphones) and distributing the training load across the workers' devices [6].

Despite these benefits, the distributed nature of FL makes it vulnerable to poisoning attacks [5, 11]. Since the server cannot control the behavior of the workers, any of them may deviate from the prescribed training protocol to attack the model by conducting either untargeted poisoning (*i.e.,* Byzantine) attacks [4,38] or targeted poisoning attacks [2,3,12]. In the former attacks, the attacker aims at degrading the model's overall performance; in the latter attacks, he aims at causing the global model to incorrectly classify some attacker-chosen inputs. Furthermore, poisoning attacks can be performed in two ways: model poisoning [2, 4, 38] or data poisoning [3, 12, 34]. In model poisoning, the attackers maliciously manipulate their local model parameters before sending them to the server. In data poisoning, they inject fabricated or falsified data examples into their training data before local model training. Both attacks result in poisoned updates being uploaded to the server, in order to prevent the global model from converging or to bias it. Compared to untargeted poisoning [4, 38], targeted attacks [2, 3, 12, 34, 35, 39] are more serious, given their stealthy nature and severe security implications [1, 12, 32, 34].

Several defenses against poisoning attacks have been proposed, which we discuss in Section 4. However, they are either impractical [16,28], may degrade the performance of the aggregated model on the main task [2, 35] or assume specific distributions of local training data [1, 4, 8, 12, 30, 34, 40]. Specifically, [16] assumes the server to possess some data examples representing the workers' data, which is not a realistic assumption in FL; [4,8,30,34,40] assume the local data are independent and identically distributed (iid) among workers, which leads to poor performance on non-iid data [1]; [1,12] identify workers with similar updates as attackers, which leads to wrongly penalizing honest workers with similar local data [22,29]. Besides those assumptions, the dimensionality of the model is a paramount factor that affects the performance of most of the methods above: high-dimensional models are more vulnerable to targeted poisoning attacks because they cause small changes on a poisoned local update without being detected [7]. To the best of our knowledge, no contribution to the state of the art provides an effective defense against targeted poisoning attacks without being hampered by the data distribution or model dimensionality.

Contributions and plan. In this paper, we study targeted poisoning attacks by analyzing the behavior of label-flipping (LF) and backdoor (BA) attacks on DL models. We find that an unbalanced distribution of the workers' local data and a high dimensionality of the DL model make the detection of these attacks quite challenging. Moreover, we observe that attack-related last-layer neurons exhibit a different behavior from attack-unrelated last-layer neurons, which makes the last-layer gradients useful features for detecting targeted attacks. Accordingly, we propose *FL-Defender*, a method that can mitigate the attacks regardless of the model dimensionality or the distribution of the workers' data. First, we use the workers' last-layer gradients to engineer more robust discriminative features that capture the attack behavior and discard redundant information. Specifically, we compute the worker-wise angle similarity for the workers' last-layer gradients and then compress the computed similarity vectors using principal component analysis (PCA) [37] to reduce redundant information. After that, we penalize the workers' updates based on their angular deviation from the centroid of the compressed similarity vectors. Experimental results on three data sets with different DL model sizes and worker data distributions demonstrate the effectiveness of our approach at defending against the attacks. Compared with several state-of-the-art defenses, *FL-Defender* achieves better performance at retaining the accuracy of the global model on the main task, reducing the attack success rate and causing minimal computational overhead on the server.

The rest of this paper is organized as follows. Section 2 introduces preliminary notions. Section 3 formalizes the attacks and the threat model being considered. Section 4 discusses countermeasures for poisoning attacks in FL. Section 5 analyzes the behavior of label-flipping and backdoor attacks,

and shows the robustness of the engineered features. Section 6 presents the methodology of the proposed defense. Section 7 details the experimental setup, and reports the obtained results. Finally, conclusions and future research lines are gathered in Section 8.

2 Preliminaries

Deep neural network-based classifiers. A deep neural network (DNN) is a function $F(x)$, obtained by composing L functions $f^l, l \in [1, L]$, that transforms an input x to a predicted output $\hat{y}$. Each f^l is a layer that is parameterized by a weight matrix w^l, a bias vector b^l and an activation function σ^l. f^l takes as input the output of the previous layer f^{l-1}. The output of f^l on an input x is computed as $f^l(x) = \sigma^l(w^l \cdot x + b^l)$. Therefore, a DNN can be formulated as

$$F(x) = \sigma^L(w^L \cdot \sigma^{L-1}(w^{L-1} \dots \sigma^1(w^1 \cdot x + b^1) \cdots + b^{L-1}) + b^L).$$

A DNN-based classifier consists of a feature extractor and a classifier [20, 25]. The classifier makes the final classification decision based on the extracted features and usually consists of one or more fully connected layers where the last layer contains $|\mathcal{C}|$ neurons with $\mathcal{C}$ being the set of all possible class values. The output layer's vector $o \in \mathbb{R}^{|\mathcal{C}|}$ is usually passed to the softmax function that converts it to a vector p of probabilities, which is called the vector of confidence scores.

多层感知机的模型的形式，在最后一层的全连接。

In this paper, we use predictive DNNs as $|\mathcal{C}|$-class classifiers, where the index of the highest confidence score in p is considered the final predicted class $\hat{y}$. Also, we analyze the last layers' gradients of DNNs to filter out poisoned updates resulting from LF and backdoor attacks.

Federated learning. In FL, an aggregator server and K workers cooperatively build a shared global model. The server starts by randomly initializing the global model W^t. Then, at each training round, the server selects a subset of workers S of size $C \cdot K \geq 1$ where K is the total number of workers in the system, and C is the fraction of workers that are selected in the training round. After that, the server distributes the current global model W^t to all workers in S. Besides the global model, the server sends a set of hyper-parameters to be used at the workers' side to train their model locally: number of local epochs E, local batch size BS and learning rate η. After receiving the new shared model W^t, each worker divides her local data into batches of size BS and performs E local training epochs of stochastic gradient descent (SGD). Finally, workers upload their updated local models $W_{(k)}^{t+1}$ to the server, which then aggregates them to obtain the new global model W^{t+1}. The federated averaging algorithm (*FedAvg*) [24] is usually employed to perform the aggregation as

$$W^{t+1} = \sum_{k=1}^{K} \frac{n_{(k)}}{n} W_{(k)}^{t+1},$$

where $n_{(k)}$ is the number of data points locally held by worker k and n is the total number of data points locally held by the K workers, that is, $n = \sum_{k=1}^{K} n_{(k)}$.

3 Attacks and threat model

As mentioned above, we focus on the LF and BA attacks, two widely used targeted attacks in the FL literature. In the LF attack [3, 12], each attacker poisons his training data set D_k as follows: for all examples in D_k with a class label c_{src}, change their class label to c_{target}. An example is changing the

标签翻转攻击是把源类的标签转化成目标类的标签

labels of "fraudulent" activities to "non-fraudulent". In the BA attack [13], the attacker poisons his training data by embedding a specific pattern (the backdoor) into training examples with specific features and assigns them a target class label of his choice. The pattern acts as a trigger for the global model to output the desired target label for the backdoored examples. As a famous example, the attacker could put a small sticker on a "stop traffic sign" and change its label to "speed limit". After poisoning their data, the attacker trains his local models on the poisoned data, with the same training settings as the honest workers, and uploads the resulting poisoned updates to the server to be aggregated into the global model.

Assumptions on training data distribution. Since the local data sets of the workers may come from heterogeneous sources [6, 36], they may be either identically distributed (iid) or non-iid. In the iid setting, each worker holds local data representing the whole distribution. In the non-iid setting, the distributions of the workers' local data sets can be different in terms of the classes represented in the data and/or the number of samples each worker holds for each class. Consequently, each worker may have local data with i) all the classes being present in a similar proportion as in the other workers' local data (iid setting), ii) some classes being present in a different proportion (non-iid setting).

Threat model. We consider a number of attackers $K' \leq K/5$, that is, no more than 20% of the K workers in the system. Although some works in the literature assume larger percentages of attackers, finding more than 20% of attackers in real-world FL scenarios is unlikely. For example, with millions of users [9] in Gboard [14], controlling a small percentage of user devices requires the attacker(s) to compromise a large number of devices, which demands huge effort and resources and is therefore impractical. Furthermore, we assume the FL server to be honest and not compromised, and the attackers to have no control over the aggregator or the honest workers. The attacker's goal for the LF attack is to cause the learned global model to classify the source class examples into the target class at test time. The attacker's goal for the BA attack is to fool the global model into falsely predicting the attacker's chosen class for any target example carrying the backdoor pattern, while maintaining the benign model performance on non-backdoored examples.

4 Related work

Existing methods to counter poisoning attacks in FL are based on one of the following principles.

Evaluation metrics. An update is penalized as being probably bad if it degrades an evaluation metric of the global model, *e.g.* its accuracy. [16, 28] use a validation set on the server to compute the loss caused by each local update and then keep or discard an update based on its computed loss. However, this is impractical in FL because the server does not have access to the workers' data. In addition, this approach cannot properly detect backdoor attacks because these attacks have little to no impact on the model's performance on the main task.

Clustering updates. Updates are clustered into two separate groups, where the smaller group contains all the bad updates, that are subsequently disregarded in the model learning process. The Auror [30] and the multi-Krum (MKrum) [4] methods assume that the workers' data are iid, which explains their poor performance on non-iid data [1], where they incur high false positive and false negative rates.

Worker behavior. This approach assumes the attackers' behavior is reflected in their updates, making them more similar to each other than honest workers' updates. Hence, updates are penalized based on their similarity. For example, FoolsGold [12] and CONTRA [1] limit the contributions of similar updates by reducing their learning rates or preventing their originators from being selected.

However, these methods also penalize similar good updates, which results in significant drops in the model performance [22, 29] when good updates are similar.

Update aggregation. This approach uses robust update aggregation rules, such as the median [40], the trimmed mean [40] or the repeated median [31]. However, the performance of these rules deteriorates on non-iid data because they discard most of the information at the time of update aggregation. Also, their estimation error scales up proportionally to the square root of the model size [7].

Differential privacy (DP). Methods under this approach [2, 33] clip individual update parameters to a maximum threshold and add random noise to the parameters to reduce the impact of potentially poisoned updates on the aggregated global model. However, there is a trade-off between adding noise to mitigate the attacks and maintaining the benign performance of the aggregated model on the main task [2,35]. Also, DP-based methods consider only mitigating backdoor attacks and are not designed to counter label-flipping attacks.

Several works propose to analyze specific parts of the updates to counter poisoning attacks. [17] proposes analyzing the last layer's biases. However, it assumes the local data are iid and form two separate clusters. FoolsGold [12] analyzes the last layer's weights to counter targeted poisoning attacks. However, as mentioned above, it performs poorly when good updates are similar because it considers them to be bad. To counter LF attacks, [34] uses PCA to analyze the weights associated with *the possibly attacked source class* and excludes potential bad updates that differ from the majority of updates in those weights. However, the method is evaluated only for LF attacks under the iid setting, and it requires prior knowledge on the possible source class.

The methods just cited share the shortcomings of (i) making assumptions on the distributions of the workers' data and (ii) not providing analytical or empirical evidence of why focusing on specific parts of the updates contributes towards defending against the attacks. In contrast, we provide comprehensive conceptual and empirical analyses that explain why focusing on the last-layer gradients is especially useful to defend against targeted poisoning attacks. Also, we propose a more robust method that can mitigate the attacks without being limited by the distribution of the workers' data or the dimension of the models in use.

5 Analysing targeted attacks against FL

This section is key in our work. We study the behavior of label-flipping and backdoor attacks to find robust discriminative features that can detect such attacks.

Let us consider an FL classification task where each local model is trained with the cross-entropy loss over one-hot encoded labels as follows. First, the activation vector o of the last layer neurons (a.k.a. logits) is fed into the softmax function to compute the vector p of probabilities as follows:

$$p_k = \frac{e^{o_k}}{\sum_{j=1}^{|\mathcal{C}|} e^{o_j}}, \quad k = 1, \ldots, |\mathcal{C}|.$$

Then, the loss is computed as

$$\mathcal{L}(y, p) = -\sum_{k=1}^{|\mathcal{C}|} y_k \log(p_k),$$

where $y = (y_1, y_2, \ldots, y_{|\mathcal{C}|})$ is the corresponding one-hot encoded true label and p_k denotes the confidence score predicted for the k^{th} class. After that, the gradient of the loss w.r.t. the output o_i

暂时还没有推导出来，怎么样得出的结论，先暂时放着。

of the i^{th} neuron (a.k.a the i^{th} neuron error) in the output layer is computed as

$$\delta_i = \frac{\partial \mathcal{L}(y,p)}{\partial o_i} = p_i - y_i. \tag{1}$$

Note that δ_i will always be in the interval $[0,1]$ when $y_i = 0$ (for the wrong class neuron), while it will always be in the interval $[-1,0]$ when $y_i = 1$ (for the true class neuron).

The gradient ∇b_i^L w.r.t. the bias b_i^L connected to the i^{th} neuron in the output layer can be written as

$$\nabla b_i^L = \delta_i \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}, \tag{2}$$

where a^{L-1} is the activation output of the previous layer. Likewise, the gradient ∇w_i^L w.r.t. the weights vector w_i^L connected to the i^{th} neuron in the output layer is

$$\nabla w_i^L = \delta_i a^{L-1} \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}. \tag{3}$$

From Equations (2) and (3), we can notice that δ_i directly and highly impacts on the gradients of the output layer's weights and biases.

Behavior of label-flipping attacks. In FL, a label-flipping attacker always tries to minimize $p_{c_{src}}$ for any example in his training data, including examples that belong to class c_{src}. On the other side, he always tries to maximize $p_{c_{target}}$ for examples that belong to c_{src} during model training. Since this goes in the opposite direction of the objective of honest workers for examples in c_{src}, the attack will entail substantial alteration of $\delta_{c_{src}}$ and $\delta_{c_{target}}$, as it can be seen from Expression (1). In turn, from Expressions (2) and (3), it follows that altering $\delta_{c_{src}}$ and $\delta_{c_{target}}$ directly alters the biases and weights corresponding to the output neurons of c_{src} and c_{target} during the training of the attacker's local model. Hence, the impact of the attack can be expected to show in the gradients of the last-layer neurons corresponding to c_{src} and c_{target}. However, the last layer is likely to contain other neurons unrelated to the attack where both the attacker and the honest workers share the same objectives, which makes the attack harder to spot. Considering all layers is still worse, because in the layers different from the last one the impact of the attack will be even less perceptible (as it will be mixed with more unrelated parameters). Moreover, there are two other factors that increase the difficulty of detecting the attack by analyzing the update as a whole: i) the early layers usually extract common features that are not class-specific [27] and ii) in general, most parameters in DL models are redundant [10]. That causes the magnitudes and angles of the bad and good updates to be similar, which makes models with large dimensionality an ideal environment for a successful label-flipping attack.

在最后一层的参数中还包括一些

Behavior of backdoor attacks. Backdoor attacks might be viewed as a particular case of label-flipping attacks because the attacker flips the label of a training example when it contains a specific feature or pattern, whereas he retains the correct label when the example does not contain the pattern (clean example). However, since the global model will correctly learn from a majority of honest workers and, in the clean examples, from the attackers as well, the received global model will probably overlook the backdoor pattern and assign the correct classes to backdoored examples, especially in the early training iterations. This will prompt the attackers to try to minimize $p_{c_{src}}$ and maximize $p_{c_{target}}$ for the backdoored examples, which can be expected to stand out in the magnitudes and the directions of the gradients contributed by the attackers. On the other hand,

since the attackers also try to maximize $p_{c_{src}}$ for their clean examples, the impact of the backdoor attacks is expected to be stealthier compared to that of LF attacks even when looking at the last-layer gradients.

结论性：对于有针对性的攻击，关注分析最后一层的梯度比分析所有层的梯度更好。因为对于有针对性的攻击而言，最后一层更能体现攻击的相关性。

Engineering more robust features to detect attacks. From the above analysis, it is clear that focusing on analyzing last-layer gradients is more helpful to detect targeted attacks than analyzing all layers. Nevertheless, the presence of a large number of attack-unrelated gradients in the last layer may still render attack detection difficult. Getting rid of those redundant and unrelated gradient features could help obtain more robust discriminatory features for targeted attacks. Since the impact of the targeted attacks is directly reflected in the directions of gradients of attack-related neurons, comparing the difference in directions between the gradients of a good update and a poisoned update can be expected to better capture the attack's behavior. If we look at the angular similarity of the workers' last-layer gradients, the similarity values between good and poisoned updates are expected to display unique characteristics in the computed similarity matrix. PCA can be used to capture those unique characteristics from the matrix and reduce redundant features.

Empirical analysis. To empirically validate our previous conceptual discussion, we used 20 local updates resulting from simulating an FL scenario under the LF and BA attacks with each of the CIFAR10-IID and CIFAR10-non-IID benchmarks, where 4 updates (that is, 20%) were poisoned. In these two benchmarks, the ResNet18 [15] architecture, which contains about $11M$ parameters was used. In addition, training data were randomly and uniformly distributed among workers in CIFAR-IID, while we adopted a Dirichlet distribution [26] with $\alpha = 1$ to generate non-iid data for the 20 workers in the CIFAR10-non-IID. The details of the experimental setup are given in Section 7.1.

Then, for each benchmark and attack scenario, we computed the following:

- The first two principal components (PCs) of the all-layer gradients for each local update. Then, we computed the centroid (CL) of the first two PCs for the 20 local updates. After that, we computed the angle between CL and every pair of PCs for each update.
- The centroid (CL) of the last-layer gradients for the 20 local updates and the angle between CL and each last-layer gradient for each update.
- The first two PCs of the last-layer gradients for each local update. Then, we computed the centroid (CL) of the first two PCs of the 20 last-layer gradients. After that, we computed the angle between CL and every pair of PCs for each last-layer gradient.
- The cosine similarity for the 20 workers' last-layer gradients. Then, we computed the first two PCs of the similarity matrix. After that, we computed the centroid (CL) of the first two PCs of the 20 similarity vectors. Finally, we computed the angle between CL and every pair of PCs of each similarity vector.

Once the above computations were completed, we visualized the magnitude of each input, and the angle between the input and its corresponding centroid.

Fig. 1 shows the visualized vectors for the CIFAR-IID benchmark. For the LF attack, we can see that analyzing the first two PCs of the all-layer gradients (All) led to poisoned updates and good updates with very similar magnitudes and angular deviation from the centroid, which made it quite challenging to tell them apart. The same applies to analyzing the last-layer gradients (Last). On the other hand, analyzing the first two PCs of the last-layer gradients (Last-PCA) led to an apparent separation between good and bad updates. This also applied to analyzing the engineered features (Engineered). For the BA attack, the results were similar with the difference that our engineered

features allowed better separation than Last-PCA. This confirms our conceptual discussion that redundant and attack-unrelated gradients make the attacks stealthier.

不是特别的理解这里的Last-PCA和Engineered的区别在哪里？暂时先理解为最后一层的梯度中进行了一些处理得出来的鲁棒特征。区别在于：1、最后一层的梯度直接进行PCA；
2、提取工人本人更新的最后一层的梯度，计算余弦相似度，然后在使用PCA压缩计算的相似度向量。

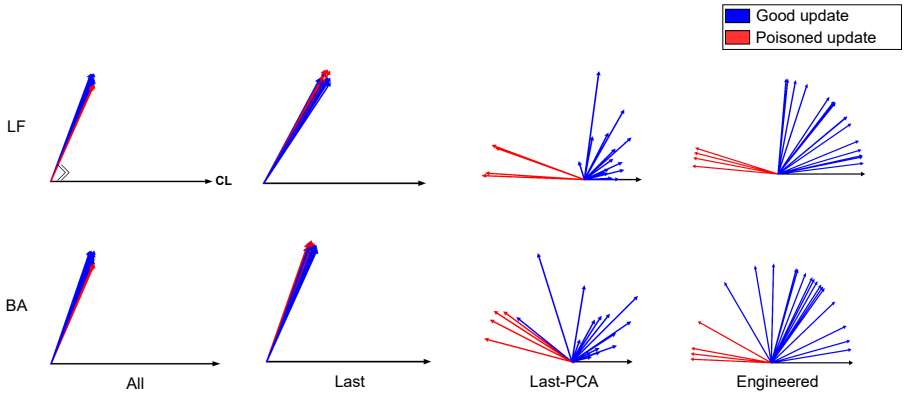

Fig. 1. Deviation of CIFAR10-IID gradient features from the centroid

Fig. 2 shows the visualized vectors for the CIFAR-non-IID benchmark, where the data were non-iid among workers. For the LF attack, we can see that only our engineered features provided robust discrimination between good and bad updates. For the BA attack, even if our engineered features allowed better separation, it was challenging to tell updates apart. This is because of the impact of the non-iidness and also because BA attacks are stealthier than LF attacks. This again confirms our intuitions and shows that our engineered features are more useful than the alternatives to detect targeted attacks.

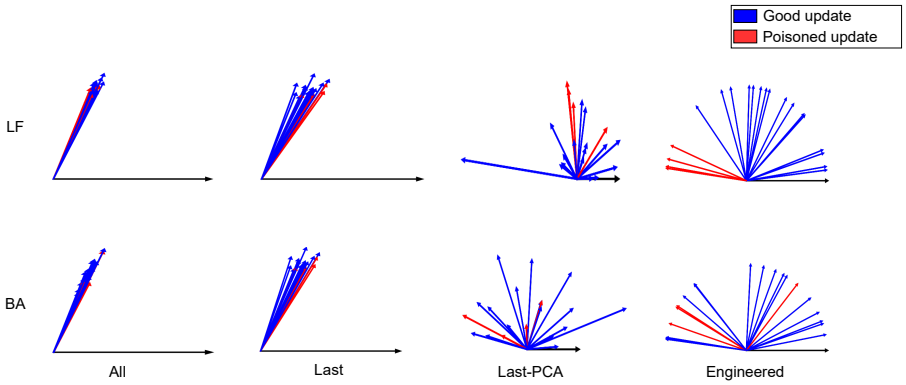

Fig. 2. Deviation of CIFAR10-non-IID gradient features from the centroid

可以看出来最后一层的梯度可以作为

We were able to conclude from these analytical and empirical explorations that a high model dimensionality and the distribution of the workers' training data highly impact on the ability to discriminate between good updates and updates poisoned by targeted attacks. Fortunately, it also became evident that we can use the last-layer gradients to engineer more robust discriminative features for attack detection.

6 FL-Defender design

In this section, we present the design of FL-Defender, our proposed defense against FL targeted poisoning attacks. Our aims are: 1) to prevent attackers from achieving their goals by mitigating the impact of their poisoned updates on global model, 2) to maintain the global model performance on the main task, 3) to stay robust against the attacks regardless of the model size or the workers' data distribution, and 4) to avoid substantially increasing the computational cost at the server. According to the lessons learned in the previous section, our defense first extracts the last-layer gradients of the workers' local updates, computes the worker-wise cosine similarity and then compresses the computed similarity vectors using PCA to reduce redundant information and extract more robust features. After that, it computes the centroid of the compressed similarity vectors and computes the cosine similarity values between the centroid and each compressed vector. The similarity values are accumulated during training and used, after re-scaling them, to re-weight the workers updates in the global model aggregation.

We formalize our method in Algorithm 1. The aggregator server A starts a federated learning task by initializing the global model W^0 and the history vector H^0 that is used to accumulate the similarities between the directions of the workers' engineered features and their centroid.

Then, in every training iteration t, A selects a random subset S of m workers and sends the current global model W^t to the m selected workers. Each worker $k \in S$ locally trains W^t on her data D_k and sends her local update W_k^{t+1} back to A. Once A receives the m local updates, it separates the gradients of the last layers to obtain the set $\{\nabla L_k^{t+1} | k \in S\}$.

Cosine similarity. We compute the cosine similarities among gradients of the last layers to capture the discrepancy between the gradients from the honest workers and those from the attackers. This discrepancy is caused by their contradicting objectives. The cosine similarity between two gradients ∇_i and ∇_j is defined as:

$$cs(\nabla_i, \nabla_j) = \cos\varphi = \frac{\nabla_i \cdot \nabla_j}{||\nabla_i|| \cdot ||\nabla_j||}.$$

This way, if ∇_i and ∇_j lie in the same direction, their cosine similarity will be 1, and their similarity value will decrease as their directions differ more. The cosine similarity measures the angular similarity among gradients and it is more robust than the Euclidean distance because, even if the attackers scale their model update gradients to avoid detection, they need to keep their directions to achieve their objectives.

Compressing similarity vectors. Since the number of workers is expected to be large in FL, we use PCA [37], with two principal components, to compress the computed similarity matrix and reduce the attackers' chance to hide their impact on the high-dimensional similarity matrix. PCA returns a compact representation of a high-dimensional input by projecting it onto a subspace of lower dimension so that the unique characteristics of the input are reduced to the subspace.

Similarity between compressed vectors and their centroid. After compressing the workers' similarity vectors into a pair of PCs, we aggregate the latter component-wise using the median to obtain their centroid $CL \in \mathbb{R}^2$. Since we assume the majority of the workers ($\geq 80\%$) to be honest, the centroid is expected to fall into the heart of a majority of good components. After that, we compute the cosine similarity between the centroid and each worker's pair of PCs. Good components are expected to have a similar direction to the centroid and thus have similarity values close to 1. On the other hand, poisoned components are expected to be farther from the centroid with values closer to -1.

Algorithm 1: FL-Defender: Combating targeted attacks in FL

Input: K, C, BS, E, η, T
Output: W^T, the global model after T training rounds

A initializes W^0, $H^0 = \{H_k^0 = 0\}_{k=1}^K$;
for *each round* $t \in [0, T-1]$ **do**
  $m \leftarrow \max(C \cdot K, 1)$;
  $S \leftarrow$ random set of m workers;
  A sends W^t to all workers in S;
  for *each worker* $k \in S$ ***in parallel*** **do**
    $W_k^{t+1} \leftarrow$ `WORKER_UPDATE`(k, W^t); // A `sends` W^t `to each worker` k `who trains` W^t `using her data` D_k `locally, and sends her local update` W_k^{t+1} `back to the aggregator.`
  end
  Let $(\nabla_1, \ldots, \nabla_i, \ldots, \nabla_m)$ be the gradients of the last layers of $\{W_k^{t+1} | k \in S\}$;
  $(cs_{1,1}, \ldots, cs_{i,j}, \ldots, cs_{m,m}) \leftarrow$ `COSINE_SIMILARITY`$(\nabla_1, \ldots, \nabla_i, \ldots, \nabla_m)$; // $\forall i, j \in (1, \ldots, m)$, $cs_{i,j}$ `is the cosine similarity between` ∇_i `and` ∇_j.
  $((p_1^1, p_1^2), \ldots, (p_i^1, p_i^2), \ldots, (p_m^1, p_m^2)) \leftarrow$ `PCA`$((cs_{1,1}, \ldots, cs_{m,m}), components = 2)$; // (p_i^1, p_i^2) `are the first two PCs of` $(cs_{i,1}, \ldots, cs_{i,m})$.
  $CL \leftarrow$`Median`$(((p_1^1, p_1^2), \ldots, (p_i^1, p_i^2), \ldots, (p_m^1, p_m^2)))$; // CL `is the centroid of the compressed similarity vectors.`
  Let cs_i be the cosine similarity between (p_i^1, p_i^2) and CL;
  for *each worker* $k \in S$ **do**
    $cs_k^t \leftarrow$ `Assign`$(cs_1, \ldots, cs_i, \ldots, cs_m)$; `// Assign the computed cosine similarities to their corresponding workers.`
    $H_k^t = H_k^{t-1} + cs_k^t$; `// Accumulate the similarities of the worker to the centroid.`
  end
  $Q1 \leftarrow$ `FIRST_QUARTILE`(H^t);
  $\gamma = H^t - Q1$; `// Subtract Q1 from every entry in` H^t`. Since attackers are expected to be below Q1, this will make their trust values in` γ `negative.`
  for *each worker* $k \in (1, \ldots, K)$ **do**
    if $\gamma_k < 0$ **then**
      $\gamma_k = 0$; `// Attacker trusts are brought to 0 to neutralize them in the aggregation.`
  end
  $\gamma = \gamma / max_k(\gamma)$; `// Normalize trust in workers updates to 0-1 range.`
  A aggregates $W^{t+1} \leftarrow \frac{1}{\sum_{k \in S} \gamma_k} \sum_{k \in S} \gamma_k W_k^{t+1}$.
end

Update history and compute updates trust scores. We use the similarity values with the centroid to update the similarity history vector H for the selected m workers. This guarantees that, as the training evolves, the closest workers to the centroid have larger and larger values than the farthest workers. Since we assume the centroid falls amid the honest workers, this guarantees that honest workers have larger accumulated similarity values than attackers. After updating the similarity history vector H, we compute the first quartile $Q1$ for the values in H and then subtract it from the similarity value accumulated by every worker. That is, we shift every similarity value in H to the left by $Q1$ and we assign the shifted similarities to the workers' trust scores vector γ.

Since the accumulated similarities of attackers are low, they are likely to be below $Q1$ and hence they become negative after the shift. After that, we set negative trust scores in γ to 0, in order to neutralize attackers when using trust scores as weights in the final aggregation (see below). Finally, we normalize the scores in γ to be in the range [0, 1] by dividing them by their maximum value.

Re-weighting and aggregating updates. In the final step of Algorithm 1, the server uses the trust scores in γ to re-weight the corresponding local updates and aggregates the global model using the re-weighted local updates. Note that, since $\frac{1}{\sum_{k \in S} \gamma_k} \sum \gamma_k = 1$, the convergence of the proposed aggregation procedure at the server side is guaranteed as long as *FedAvg* converges.

7 FL-Defender evaluation

In this section we compare the performance of our method with that of several state-of-the-art countermeasures against poisoning attacks.

7.1 Experimental setup

We implemented our experiments using the PyTorch framework on an AMD Ryzen 5 3600 6-core CPU with 32 GB RAM, an NVIDIA GTX 1660 GPU, and Windows 10 OS. For reproducibility, our code and data are available at `https://github.com/anonymized30/FL-Defender`.

Data sets and models. We used the following data sets and models:

- MNIST. It contains $70K$ handwritten digit images from 0 to 9 [21]. The images are divided into a training set ($60K$ examples) and a testing set ($10K$ examples). We used a two-layer convolutional neural network (CNN) with two fully connected layers on this data set (number of model parameters $\approx 22K$).
- CIFAR10. It consists of $60K$ colored images of 10 different classes [19]. The data set is divided into $50K$ training examples and $10K$ testing examples. We used the ResNet18 CNN model [15] with one fully connected layer on this data set (number of parameters $\approx 11M$).
- IMDB. Specifically, we used the IMDB Large Movie Review data set [23] for binary sentiment classification. The data set is a collection of $50K$ movie reviews and their corresponding sentiment binary labels (either positive or negative). We divided the data set into $40K$ training examples and $10K$ testing examples. We used a Bidirectional Long/Short-Term Memory (BiLSTM) model with an embedding layer that maps each word to a 100-dimensional vector. The model ends with a fully connected layer followed by a sigmoid function to produce the final predicted sentiment for an input review (number of parameters $\approx 12M$).

Data distribution and training. We defined the following benchmarks by distributing the data from the data sets above among the participating workers in the following way:

- MNIST-non-IID. We adopted a Dirichlet distribution [26] with a hyperparameter $\alpha = 1$ to generate *non-iid* data for 20 participating workers. The CNN model was trained during 200 iterations. In each iteration, the FL server asked the workers to train their models for 3 local epochs with a local batch size 64. The participants used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with learning rate = 0.01 and momentum = 0.9 to train their models.

- CIFAR10-IID. We randomly and uniformly divided the CIFAR10 training data among 20 workers. The ResNet18 model was trained during 100 iterations. In each iteration, the FL server asked the 20 workers to train the model for 3 local epochs with a local batch size 32. The workers used the cross-entropy loss function and the SGD optimizer with learning rate = 0.01 and momentum = 0.9.
- CIFAR10-non-IID. We took a Dirichlet distribution with a hyperparameter $\alpha = 1$ to generate *non-iid* data for 20 participating workers. The training settings were the same as in the CIFAR10-IID.
- IMDB. We randomly and uniformly split the $40K$ training examples among 20 workers. The BiLSTM was trained during 50 iterations. In each iteration, the FL server asked the 20 workers to train the model for 1 local epoch with a local batch size 32. The workers used the binary cross-entropy with logit loss function and the *Adam* optimizer with learning rate = 0.001.

Attack scenarios. i) Label-flipping attacks. In the CIFAR10 experiments, the attackers flipped the examples with the label *Dog* to *Cat* before training their local models. For IMDB, the attackers flipped the examples with the label *positive* to *negative*. ii) Backdoor attacks. In the CIFAR10 benchmarks, the attackers embedded a 3×3 square with white pixels in the bottom-right corner of the examples belonging to class *Car* and changed its label to class *Plane* before training their local models. In the MNIST-non-IID, the attackers embedded the same pattern in the examples belonging to class *9* and changed its label to class *0*. In all the experiments, the number of attackers K' ranged in $\{0, 2, 4\}$, which corresponds to a ratio of attackers in $\{0\%, 10\%, 20\%\}$.

Evaluation metrics. We used the following evaluation metrics on the test set examples to assess the impact of the attacks on the learned model and the performance of the proposed method w.r.t. the state of the art:

- *Test error (TE)*. This is the error resulting from the loss functions used in training. The lower the test error, the more robust the method is against the attack.
- *Overall accuracy (All-Acc)*. This is the number of correct predictions divided by the total number of predictions.
- *Source class accuracy (Src-Acc)*. We evaluated the accuracy for the subset of test examples belonging to the source class. Note that one may achieve a good overall accuracy while degrading the accuracy of the source class.
- *Attack success rate (ASR)*. It is defined as the proportion of targeted examples (with the source label or the backdoor pattern) that are incorrectly classified into the label desired by the attacker.

An effective defense against the attacks needs to retain the benign performance of the global model on the main task while reducing ASR.

7.2 Results

We evaluated the robustness of our defense against the LF and BA attacks and compared it with several countermeasures discussed in Section 4: median [40], trimmed mean (TMean) [40], multi-Krum (MKrum) [4] and FoolsGold (FGold) [12]. We also compared with the standard FedAvg [24] aggregation method (that is not meant counter security attacks). We report the average results of the last 10 training rounds to ensure a fair comparison among methods.

Robustness against label-flipping attacks. Table 1 reports the results for the LF attacks. For CIFAR-IID and when no attack is performed, our method achieved comparable performance to

FedAvg for the test error and the overall accuracy. On the other hand, in the presence of attacks, in general our method achieved the highest source class accuracy and the lowest attack success rate, whereas FoolsGold ranked second. Mkrum achieved the worst performance because it considers all layers, which caused a lot of false positives and false negatives. Note that the theoretical upper bound on the number of attackers MKrum [4] can resist is $K' = (K/2) - 2$ which corresponds to $K' = 8$ in our setting. The performance of the rest of the methods (FedAvg, Median and TMean) was diminished with regard to the protection of the source class, even though the data were iid. The reason was the large model size.

Looking at results for CIFAR10-non-IID, we can see the influence of the data distribution and the model size on the performance of the methods. However, thanks to the robust engineered features, our method preserved the global model performance while preventing the attackers from performing successful label-flipping attack.

For the IMDB benchmark, the performance of our method was almost the same as FoolsGold, which scored best. Both methods outperformed the other methods by a large margin in providing adequate and simultaneous protection for all the metrics. FoolsGold performed well in this benchmark because it is its ideal setting: updates for honest workers were somewhat different due to the different reviews they gave, while updates for attackers became very close to each other because they shared the same objective. In addition, there was no redundant information from other classes because the task was binary classification.

Table 1. Robustness against label-flipping attacks

Benchmark		CIFAR10-IID						CIFAR10-non-IID						IMDB					
K'/K	Method	FedAvg	Median	TMean	MKrum	FGold	Ours	FedAvg	Median	TMean	MKrum	FGold	Ours	FedAvg	Median	TMean	MKrum	FGold	Ours
0/20	TE	**0.80**	**0.80**	0.81	0.83	**0.80**	0.81	**0.85**	0.95	0.89	0.90	0.98	0.89	0.28	0.27	0.28	0.28	0.28	0.28
	All-Acc%	76.9	76.35	76.91	76.83	**77.24**	76.81	**75.88**	74.40	74.86	74.18	73.93	74.96	88.55	**88.75**	88.55	88.61	88.73	88.56
	Src-Acc%	63.6	65.30	66.50	65.70	**67.60**	**67.60**	**66.70**	66.10	66.10	65.90	52.90	66.20	85.88	86.12	85.88	86.12	86.16	86.1
	ASR%	15.60	14.40	13.50	13.40	**12.70**	13.10	14.90	13.60	14.80	15.50	21.40	**11.8**	14.12	13.88	14.12	13.88	13.84	13.87
2/20	TE	0.80	**0.78**	**0.79**	0.84	0.84	0.82	0.93	0.92	0.97	1.04	0.92	**0.90**	0.40	0.34	0.37	0.45	**0.29**	**0.29**
	All-Acc%	76.96	76.50	76.47	**77.04**	76.87	76.17	**74.76**	74.12	74.62	71.60	74.08	74.46	81.52	84.21	82.94	79.57	**88.66**	88.24
	Src-Acc%	55.20	55.80	57.90	53.90	65.60	**65.70**	48.40	51.70	50.40	39.10	58.40	**64.9**	66.07	72.74	69.52	61.65	**86.44**	86.3
	ASR%	23.40	23.00	20.60	24.50	15.00	**12.00**	28.70	25.10	28.70	31.90	18.80	**14.6**	33.93	27.26	30.48	38.35	**13.56**	13.74
4/20	TE	0.85	**0.81**	**0.81**	0.95	0.84	0.82	0.92	0.91	0.96	1.08	0.93	**0.90**	0.63	0.49	0.53	0.85	**0.30**	0.31
	All-Acc%	75.27	76.3	75.76	75.2	**76.42**	76.10	**75.22**	74.28	74.18	69.84	74.51	74.20	72.50	77.11	75.97	64.9	**88.45**	88.03
	Src-Acc%	44.00	54.20	47.70	38.90	63.00	**65.10**	44.20	48.80	42.60	21.50	51.10	**58.10**	46.04	56.26	53.78	30.36	**86.40**	**86.40**
	ASR%	33.60	25.30	31.60	37.70	16.40	**15.00**	29.60	32.30	33.60	51.50	25.50	**16.6**	53.96	43.74	46.22	69.64	13.60	**13.56**

Robustness against backdoor attacks. Fig. 3 shows the results for the backdoor attacks. We employed FedAvg when no attacks were performed as a baseline. For MNIST-non-IID (with 4 attackers), FoolsGold and our method achieved comparable results to the baseline. FoolsGold achieved such good performance because of the low variability of the MNIST data set, which made the attackers' last-layer gradients more similar to each other. On the other hand, the attackers achieved attack success rates about 100% with the other methods. For CIFAR10-IID (with 4 attackers), our method achieved superior performance compared to the other methods, which failed to counter the BA attack. We can see that our method achieved a performance very close to the baseline. For CIFAR10-non-IID results, all methods achieved poor performance against BA attacks because of the double impact of model size and data distribution. Nevertheless, our method improved over the rest of the methods in reducing the attack success rate. Besides, it maintained the global model's benign performance on the non-attacked examples.

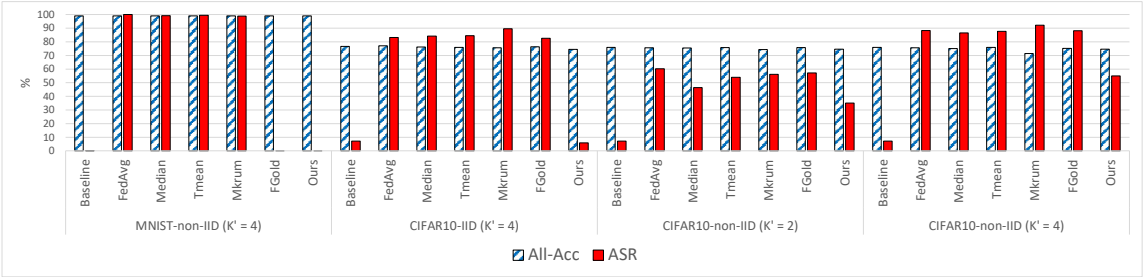

Fig. 3. Robustness against backdoor attacks

To sum up, our defense performed effectively against label-flipping attacks, while it improved over the state-of-art methods against backdoor attacks.

Runtime overhead. We measured the CPU runtime of our method and compared it with that of the other methods. Fig. 4 shows the per-iteration server runtime overhead in seconds for each method. The results show that our method and FoolsGold achieved the smallest runtime in general, excluding FedAvg, which just averages updates and is not meant to counter the attacks. Furthermore, the small runtime overhead of our method can be viewed as a good investment, given its effectiveness at combating the targeted attacks.

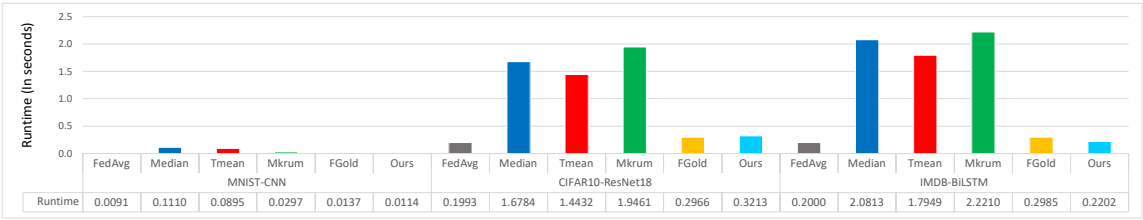

Fig. 4. CPU runtime per iteration on the server side (in seconds)

8 Conclusions and future work

In this paper, we have studied the behavior of targeted attacks against FL and we have found that robust features for attack detection can be extracted from the gradients of the last layers of deep learning models. Accordingly, we have engineered robust discriminative features for attack detection by computing the worker-wise similarities of gradient directions and then compressing them using PCA to reduce redundant information. Then, we have built on the engineered features to design FL-Defender, a novel and effective method to defend against attacks. FL-Defender re-weights the workers' local updates during the global model aggregation based on their historical deviation from the centroid of the engineered features. The empirical results show that our method performs very well at defending against label-flipping attacks regardless of the workers' data distribution or the model size. Also, it improves over the state of the art at mitigating backdoor attacks. Besides, it maintains the benign model performance on the non-attacked examples and causes minimal computational overhead on the server.

Future work directions include to deeply study the behavior of backdoor attacks on different components of DL models, in different FL settings and with different model sizes. This should yield more robust discriminative features for such dangerous and stealthy attacks.

Acknowledgments

This research was funded by the European Commission (projects H2020-871042 "SoBigData++" and H2020-101006879 "MobiDataLab"), the Government of Catalonia (ICREA Acadèmia Prizes to J.Domingo-Ferrer and D. Sánchez, and FI grant to N. Jebreel), and MCIN/AEI /10.13039/501100011033 /FEDER, UE under project PID2021-123637NB-I00 "CURLING". The authors are with the UNESCO Chair in Data Privacy, but the views in this paper are their own and are not necessarily shared by UNESCO.

References

1. Awan, S., Luo, B., Li, F.: Contra: Defending against poisoning attacks in federated learning. In: In European Symposium on Research in Computer Security (ESORICS). pp. 455–475. Springer (2021)
2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics. pp. 2938–2948. PMLR (2020)
3. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 (2012)
4. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 118–128 (2017)
5. Blanco-Justicia, A., Domingo-Ferrer, J., Martínez, S., Sánchez, D., Flanagan, A., Tan, K.E.: Achieving security and privacy in federated learning systems: Survey, research challenges and future directions. Engineering Applications of Artificial Intelligence **106**, 104468 (2021)
6. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H.B., et al.: Towards federated learning at scale: System design. arXiv preprint arXiv:1902.01046 (2019)
7. Chang, H., Shejwalkar, V., Shokri, R., Houmansadr, A.: Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. arXiv preprint arXiv:1912.11279 (2019)
8. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proceedings of the ACM on Measurement and Analysis of Computing Systems **1**(2), 1–25 (2017)
9. Davenport, C., Davenport, C.: Gboard passes one billion installs on the play store (Aug 2018), `https://www.androidpolice.com/2018/08/22/gboard-passes-one-billion-installs-play-store/`, accessed: 2022-04-2
10. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., De Freitas, N.: Predicting parameters in deep learning. arXiv preprint arXiv:1306.0543 (2013)
11. Ferrag, M.A., Friha, O., Maglaras, L., Janicke, H., Shu, L.: Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis. IEEE Access **9**, 138509–138542 (2021)
12. Fung, C., Yoon, C.J., Beschastnikh, I.: The limitations of federated learning in sybil settings. In: 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020). pp. 301–316 (2020)
13. Gu, T., Dolan-Gavitt, B., Garg, S.: Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017)

14. Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D.: Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
16. Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., Li, B.: Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 19–35. IEEE (2018)
17. Jebreel, N., Blanco-Justicia, A., Sánchez, D., Domingo-Ferrer, J.: Efficient detection of Byzantine attacks in federated learning using last layer biases. In: International Conference on Modeling Decisions for Artificial Intelligence. pp. 154–165. Springer (2020)
18. Konečnỳ, J., McMahan, B., Ramage, D.: Federated optimization: Distributed optimization beyond the datacenter. arXiv preprint arXiv:1511.03575 (2015)
19. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Communications of the ACM **60**(6), 84–90 (2017)
21. LeCun, Y., Haffner, P., Bottou, L., Bengio, Y.: Object recognition with gradient-based learning. In: Shape, contour and grouping in computer vision, pp. 319–345. Springer (1999)
22. Li, S., Ngai, E., Ye, F., Voigt, T.: Auto-weighted robust federated learning with corrupted data sources. arXiv preprint arXiv:2101.05880 (2021)
23. Maas, A., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies. pp. 142–150 (2011)
24. McMahan, B., Moore, E., Ramage, D., Hampson, S., Aguera-Arcas, B.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282. PMLR (2017)
25. Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J.: Deep learning–based text classification: A comprehensive review. ACM Computing Surveys (CSUR) **54**(3), 1–40 (2021)
26. Minka, T.: Estimating a dirichlet distribution (2000)
27. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy (SP). pp. 739–753. IEEE (2019)
28. Nelson, B., Barreno, M., Chi, F.J., Joseph, A.D., Rubinstein, B.I., Saini, U., Sutton, C., Tygar, J.D., Xia, K.: Exploiting machine learning to subvert your spam filter. LEET **8**, 1–9 (2008)
29. Nguyen, T.D., Rieger, P., Yalame, H., Möllering, H., Fereidooni, H., Marchal, S., Miettinen, M., Mirhoseini, A., Sadeghi, A.R., Schneider, T., et al.: Flguard: Secure and private federated learning. arXiv preprint arXiv:2101.02281 (2021)
30. Shen, S., Tople, S., Saxena, P.: Auror: Defending against poisoning attacks in collaborative deep learning systems. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. pp. 508–519 (2016)
31. Siegel, A.F.: Robust regression using repeated medians. Biometrika **69**(1), 242–244 (1982)
32. Steinhardt, J., Koh, P.W., Liang, P.: Certified defenses for data poisoning attacks. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 3520–3532 (2017)
33. Sun, Z., Kairouz, P., Suresh, A.T., McMahan, H.B.: Can you really backdoor federated learning? arXiv preprint arXiv:1911.07963 (2019)
34. Tolpegin, V., Truex, S., Gursoy, M.E., Liu, L.: Data poisoning attacks against federated learning systems. In: European Symposium on Research in Computer Security. pp. 480–501. Springer (2020)
35. Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.y., Lee, K., Papailiopoulos, D.: Attack of the tails: Yes, you really can backdoor federated learning. Advances in Neural Information Processing Systems **33**, 16070–16084 (2020)

36. Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., Chen, M.: In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. IEEE Network **33**(5), 156–165 (2019)
37. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1-3), 37–52 (1987)
38. Wu, Z., Ling, Q., Chen, T., Giannakis, G.B.: Federated variance-reduced stochastic gradient descent with robustness to Byzantine attacks. IEEE Transactions on Signal Processing **68**, 4583–4596 (2020)
39. Xie, C., Huang, K., Chen, P.Y., Li, B.: Dba: Distributed backdoor attacks against federated learning. In: International Conference on Learning Representations (2019)
40. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: Towards optimal statistical rates. In: International Conference on Machine Learning. pp. 5650–5659. PMLR (2018)