Möllering - Helen

Promo 2020 – Master EIT Digital Cyber Security

NEC Laboratories Europe

Heidelberg, Germany

01. 03. 2019 – 31. 08. 2019

Thwarting Semantic Backdoor Attacks in Privacy Preserving Federated Learning

Academic Supervisor: Melek Önen, Associate Professor

Industrial Supervisors: Dr. Ghassan Karame, Manager & Chief Researcher;
Dr. Giorgia Azzurra Marson, Research Scientist

Confidential thesis report

YES ☒/ NO ☐

EURECOM
Sophia Antipolis

DECLARATION POUR LE RAPPORT DE STAGE
DECLARATION FOR THE MASTER'S THESIS

Je garantis que le rapport est mon travail original et que je n'ai pas reçu d'aide extérieure. Seules les sources citées ont été utilisées dans ce projet. Les parties qui sont des citations directes ou des paraphrases sont identifiées comme telles.

I warrant, that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this report. Parts that are direct quotes or paraphrases are identified as such.

À Biot, *in Biot*
Date : 27.08.2019

Nom Prénom : Möllering, Helen
Name First Name

Signature : 

Abstract

Federated learning is a new approach for privacy-preserving machine learning which lets clients collaboratively train a shared machine-learning model while keeping all training data locally on the clients' devices and sharing only model updates to be aggregated at the server. Despite the improved data privacy, compared to other distributed solutions where (private) data is shared with other parties, current federated learning deployments are vulnerable to model-poisoning attacks that manipulate the training process so that the shared model exhibits malicious behavior. Concretely, it is possible to inject so called "backdoors" into machine learning models such that the model behaves normally on regular data, but causes a targeted misclassification on attacker-chosen samples.

In this work, we study state-of-the-art backdoor attacks and defenses on federated learning. Furthermore, we introduce the first formal definitions for all types of backdoor attacks that have been proposed so far. We implement one of these attacks, namely "semantic backdooring", and investigate its effectiveness, stealthiness, and durability in extensive experiments.

Building on these insights, we explore the solution space to protect against semantic backdoor attacks in the context of model poisoning. We define the requirements that a defense system should fulfill, and we propose three orthogonal techniques to detect malicious contributions. The first defensive technique uses statistical methods applied to fine-grained misclassification distributions to amplify the indications for poisoning. The second technique applies neural-network activation clustering to distinguish clean from infected classes. The last defensive layer is a client-driven feedback loop that allows to increase the available data for the analyses. All three techniques can be flexibly operated solo or in concert. We empirically analyze the impact of our three defense layers through extensive experiments for testing all defenses, individually and in combination, against state-of-the-art semantic backdoor attacks on federated learning with the CIFAR-10 data set. Finally, we evaluate the effectiveness of our proposals based on the aforementioned requirements, and elaborate on limitations and future work.

Résumé

Le “federated learning” est une nouvelle approche de “machine learning” qui vise à préserver la sphère privée des utilisateurs tout en permettant de former collaborativement un model partagé de “machine learning”, le tout sans jamais que les données de chacun soient divulguées aux autres participants. En effet, cette approche au “machine learning” se base uniquement sur le partage des modifications du modèle qui sont alors agrégées au niveau d'un serveur central. Malgré l'amélioration notoire de la préservation de la sphère privée, les déploiements de “federated learning” sont encore vulnérables à des attaques tel que l'empoisonnements de modèles, a contrario de modèles où les données sont partagées entre les diffèrent participants. Cette attaque peut permettre la manipulation du procéder d'entrainement de tel sorte que le modèle résultant expose un comportement malicieux, ou, en tous cas, non désirées. Concrètement, il est possible d'injecter des « backdoors » dans le modèle du “machine learning” afin que le modèle se comporte normalement avec des donnes standards, mais puisse classifier de manière erronée quand les données sont choisies par l'attaqueur.

Dans ce document, nous étudions des attaques et défenses a la pointe de la technologie du “federated learning”. De plus, nous introduisons une première définition formelle de chaque type d'attaque faisant partie des “backdoor attacks” sur les “federatetd learning”. Nous avons mise en place une de ces attaques, i.e. le “semantic backdooring”, et enquêté son efficacité, sa discrétion ainsi que sa durabilité à travers des expériences compréhensives.

En s'appuyant sur les connaissances découlant de ces expériences, nous explorons différentes solutions afin de se prémunir contre les attaques de type “semantic backdoors” dans le contexte des empoisonnements de modèles. Nous définissons également formellement les prérequis nécessaires qu'un system de défense doit satisfaire, et nous proposons trois mécanismes orthogonaux afin de détecter les contributions malicieuses. Le premier mécanisme utilise des méthodes statistiques appliquées aux distributions de classifications erronées de chaque classe afin d'amplifier les indications d'empoisonnement. Le second mécanisme regroupe les activations des neurones (“activation clustering”) afin de distinguer les classes infectées des saines. Finalement, le dernier mécanisme de défense est une boucle de rétroaction initiée par le client qui permet d'augmenter la quantité de données disponibles pour l'analyse. Les trois mécanismes présentés peuvent être

flexiblement utilisés de manière indépendant ou combiné. Nous mesurons empiriquement l'impact de chacune des barrières de défenses, individuellement ainsi que de manière combinée, ainsi que leurs efficacités contre une attaque à la pointe de l'art de type « semantic backdoor » à travers des expériences approfondies. Pour notre modèle, nous utilisons l'ensemble de données CIFAR-10. Finalement, nous évaluons l'efficacité de nos mécanismes de défenses proposés par rapport aux prérequis susmentionnées et poursuivons sur les limitations ainsi que les potentielles améliorations futures.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Motivation

The recent rise of Data Science and Artificial Intelligence is partially based on techniques like neural networks that have existed for many years [9, 10]. What leads to this recent trend is that we nowadays have access to vast amounts of data not available in the last century [11, 12]. To build reliable models, a significant amount of data is indispensable. Today's technologies, as for example social networks and IoT devices, allow to collect enormous volumes of data in real-time at a massive scale. It has been predicted that 1.7MB of data will be created per person every second in 2020 [13]. This so called "Big Data" is the basis of today's Data Science.

Analytics can support companies to accelerate their business by improving customer services and products, allows to build intelligent transport systems, and is gaining increasing attention in the health sector [14, 15]. But (personal) data is needed to build and train classifiers, as well as to evaluate their quality, and this comes with serious security and privacy concerns.

The goal of using these algorithms is to detect valuable patterns and extract new information from the data. The collected information contains personal information that needs to be protected. This is not only because of legal reasons, as for example the General Data Protection Regulation (GDPR)[1] that came into force in May 2018 and threatens with severe financial penalties if companies fail to protect the personal data, but also because of personal interests of data owners and the danger of discrimination. It follows that although end-users often might even be not aware of the sensitivity of the data that they are sharing, research has the obligation to protect their privacy. For example, the unauthorized disclosure of very sensitive information like health data can significantly impact the life of an individual.

Additionally, access to private data is prone to manipulation and misuse. This all led to research in the secure and privacy-preserving design of machine learning algorithms that aim to protect individuals whose data is used in the analysis [16, 17]. Aggarwal et al. are often claimed to be the ones who have introduced the topic with their work on constructing a decision tree classifier with perturbed data [18]. There are already several

[1] `https://gdpr-info.eu/`

approaches to protect sensitive information when publishing data sets as for example de-identification through suppression, generalization, or synthetic data generation to achieve anonymization and to protect individuals. But diverse cases that were also intensively discussed in the news show that they often fail to be a satisfying solution [19, 20, 21, 22]. In many cases, it is possible to re-identify people by the combination of different information and data sets. Other approaches, as state-of-the-art designs based on cryptography, suffer of scalability problems as they tend to be slow and costly (e.g., [23], [24]).

A new approach, called *federated learning*, tries to solve the problem by jointly training a machine learning classifier from many sources while keeping the data completely private on the clients' devices. Nevertheless, this high guarantee of privacy comes with disadvantages in terms of communication and computation costs, and also security issues. Its security concerns are the focus of this thesis.

1.2. Problem Statement

In *federated learning* all data remains locally at the client and the client only sends the model updates after a training phase to a central aggregator. This aggregator averages the updates of several participants to get a new improved global model. As also individual local updates might leak information about a client's data, an aggregation mechanism has been proposed that allows to combine clients' updates in a privacy-preserving manner before the aggregator gets access to them (to include it into its global model) [4]. This approach seems to solve the privacy problem in machine learning as there is no access to individual data.

Nevertheless, other issues arise from this scenario. Recent work [7, 25] has shown that federated learning is prone to so called *model poisoning*-attacks where malicious clients send manipulated updates that let the model perform normally (and therefore not detectable) for all samples except from specific attacker-chosen samples that are misclassified in a targeted or non-targeted way. These samples are called *backdoors*. Such attacks can, for example, be used to fool facial recognition systems, or in the case of autonomous driving, to trick the self-driving car by teaching it to misinterpret stop signs [68].

To inject a *backdoor*, malicious clients select specific images - depending on the type of attack they might also minorly manipulate them - and possibly a target class as well. They train with this poisoned data and return the poisoned model update. As the server does not know anything about the client's data and has no influence on the local training process, it cannot know if normal or manipulated data and labels were used or if the training process was somehow manipulated. Additionally, data is expected to

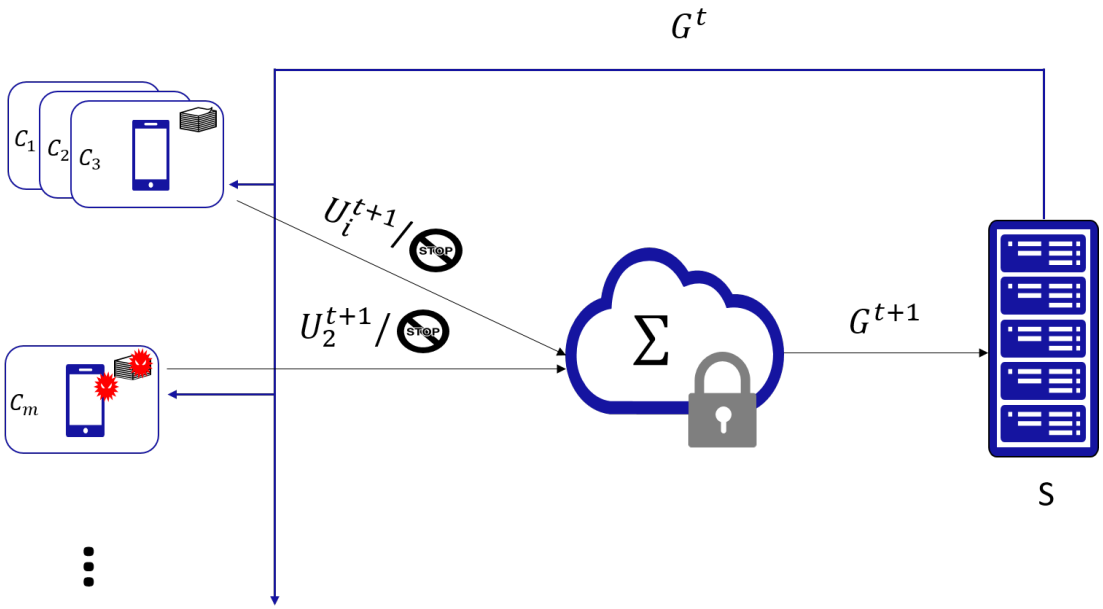


Figure 1.: Malicious clients in federated learning

be especially distributed and unbalanced in federated learning which is why comparing users' updates does not give sufficient insights to detect poisoning [7]. Furthermore, in the case of secure update aggregation, the aggregator does not even have access to the individual's update at all which is the reason why Bagdasryan et al. claim in [7] that there is no defense possible.

Figure 1 shows a federated learning iteration where a malicious client is chosen to contribute in the updating process in which it injects its manipulated model.

1.3. Contribution

In this work, we investigate federated learning by implementing a local federated learning system with different strategies for splitting the data among clients and data sets. Furthermore, we perform an intensive study of related work about attacks on federated learning and the proposed defenses, next to other machine learning techniques' security issues, attacks, and defenses. We implement semantic backdoor attacks on our implementation to evaluate their effectiveness. Finally, we design, implement, and test three defense layers that aim to thwart/reduce the problem of model poisoning in federated learning.

2. Background

In this chapter, we define necessary terms and introduce background knowledge that are used in the scope of this thesis. In Section 2.1, we start with a short introduction of general notation, before defining machine learning itself and explaining important components of it, such as the gradient descent optimization algorithm in Section 2.2. In Section 2.2.2, we present the class of machine learning algorithms called neural networks, which are in the focus of this work. Finally, in Section 2.3, we describe the recently proposed privacy-preserving way of training machine learning models called federated learning, together with the privacy and security risks that arise from it.

2.1. General Notation

For $a, b \in \mathbb{N}$, we write $[a\,..\,b] := \{x \in \mathbb{N} : a \leq x \leq b\}$. Let X be a (finite) set, and let $\mathcal{D}\colon X \to [0,1]$ be a probability distribution. We denote by $x \leftarrow_{\mathcal{D}} X$ the random sampling of an element x according to distribution $\mathcal{D}$. We write $x \leftarrow_{\$} X$ for the special case of sampling x uniformly at random.

2.2. Machine Learning

Machine learning denotes an automated learning process. Generally, it is concerned with the problem of approximating an unknown function $f : \mathcal{X} \to \mathcal{Y}$ by $f_\theta \in F_\Theta = \{f_\theta, \theta \in \Theta\}$. F_Θ denotes the family of computable functions. $\mathcal{X}$ and $\mathcal{Y}$ are the sets of instances and corresponding labels respectively [3, pp. 33-34].

There are two main approaches in machine learning: *supervised learning*, and *unsupervised learning*. Both encompass several classes of algorithms. In supervised learning, an algorithm takes a set of input/output pairs $(x_i, y_i) \in D \subseteq \mathcal{X} \times \mathcal{Y}$ to generate a mapping of new inputs x_j to labels y_j that should follow the scheme given by previously provided data. Unsupervised learning denotes the process of, given an input set $(x_i) \in D \subseteq \mathcal{X}$, finding distinguishing and common characteristics and patterns in the data such that the input elements can be grouped according to their similarities [26]. In this work, we focus on supervised learning.

Supervised learning can be split into two phases. The first phase, when an algorithm learns from given data D, is called *training phase*. The second phase uses the learned

information from the first phase to predict labels for new, non-labeled data, and is called the *inference phase*.

The *training phase* is an optimization process in which the loss/error over a data set D for a wrong output $f_\theta(x)$ compared to the true label $f(x) = y$ is minimized. In the following, we always denote a true label by y. The training is done by adapting the parameters of the function indicated by θ such that the objective function *loss* is minimized [3, pp. 35]:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in D} loss(f_\theta(x), y) \tag{1}$$

The output of the first phase is called *trained model* or *classifier* $f_{\hat{\theta}}$. In the following, we use the two expressions interchangeably. It can afterwards be used in the *inference phase* to predict labels for unknown data:

$$f_{\hat{\theta}}(x) = y, x \in D_{new} \subseteq \mathcal{X}, D_{new} \cap \{x \in \mathcal{X} : (x, *) \in D\} = \emptyset \tag{2}$$

To quantify the quality of a model for the unknown relation f the two metrics *true accuracy* and *true error* are used. *True accuracy* denotes the proportion of correctly classified samples by the model $f_{\hat{\theta}}$ as presented in Equation 3. Equation 4 defines the *true error* of a model $f_{\hat{\theta}}$ which is the proportion of inputs x that are not correctly classified.

$$acc(f_{\hat{\theta}}) := \Pr_{x \leftarrow_{\mathcal{D}} \mathcal{X}}[f_{\hat{\theta}}(x) = y] \tag{3}$$

$$err(f_{\hat{\theta}}) := \Pr_{x \leftarrow_{\mathcal{D}} \mathcal{X}}[f_{\hat{\theta}}(x) \neq y] \tag{4}$$

Because in practice the real distribution $\mathcal{D}$ of the input values x is unknown, the *empirical accuracy* is normally calculated instead of the true accuracy. It denotes the probability of a correct prediction by the trained model $f_{\hat{\theta}}$ based on a given data set $D = \{(x, y) : x \in X, y = f(x)\}$, where $X \subseteq \mathcal{X}$ [27]. For such a set $X \subseteq \mathcal{X}$ with corresponding labels in D, we denote by $X_{\checkmark}(f_{\hat{\theta}}) = \{x \in X : f_{\hat{\theta}}(x) = y\}$ the set of instances where $f_{\hat{\theta}}$ agrees with the true relation f. $X_{\text{✗}}(f_{\hat{\theta}}) = \{x \in X : f_{\hat{\theta}}(x) \neq y\}$ denotes the set of misclassified instances. If D is the labeled set associated to X, we may also write $D_{\checkmark}(f_\theta)$ and $D_{\text{✗}}(f_\theta)$ for the above mentioned sets. Therefore, the *empirical accuracy* can be defined as follows:

$$acc_D(f_{\hat{\theta}}) = \frac{|D_{\checkmark}(f_{\hat{\theta}})|}{|D|} \tag{5}$$

Additionally, the *empirical error* that is effectively equal to $1 - acc_D$ can also be used

to measure how well $f_{\hat{\theta}}$ approximates f:

$$err_D(f_{\hat{\theta}}) = \frac{|D_{\text{✗}}(f_{\hat{\theta}})|}{|D|} \tag{6}$$

Generally, it is crucial that a model generalizes and does not *overfit*. Overfitting indicates that a model performs well on training data but badly on new data [3, p. 36]. To avoid overfitting in supervised learning, the given data D can be split into two mutually exclusive sets: a training set D_{train} and a test set D_{test}. This process is called *holdout*. The test set D_{test} is reserved for assessing the quality of the prediction after the model has been fully trained and is never used in the training process, whereas D_{train} is used to derive the model.

2.2.1. Gradient Descent

As stated before, the training phase of a machine learning algorithm describes an optimization process which minimizes an *objective function* indicated above by $loss$. We denote the model of every iteration t with f_{θ_t}. The idea is that the model f_{θ_t} changes in each iteration in such a way that it makes as less as possible wrong predictions. The wrong predictions are encoded in the objective function $loss$, consequently the function is minimized.

A typical strategy in supervised learning to solve such optimization problems is the *gradient descent* algorithm. It is an iterative algorithm that changes the parameters θ of the current model in each round in the direction of the steepest descent of the objective function. It can only be applied in a continuous space and it requires $loss$ and the f_θ's to be differentiable [8, p. 80-84].

After a random initialization of all trainable parameters θ_t^i, $i \in \{0, ..., num_{parameters}\}$, $t \in \{0, ..., num_{iterations}\}$ of a model f_{θ_t}, the *gradient* $\nabla loss_t$ of the objective function $loss$ is determined based on the training data D_{train} in each iteration t. The gradient contains the partial derivatives $\frac{\partial loss}{\partial \theta_t^i}$ of all parameters θ_t^i of the model f_{θ_t}. Derivatives point towards the direction of the steepest ascent. Therefore, moving into the opposite direction will give the optimal update at iteration t for minimizing the loss [8, p. 80-84]:

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{\partial loss}{\partial \theta_t} \tag{7}$$

η is called *learning rate*. Because it influences how fast a model changes into the direction of the gradient it needs to be carefully chosen to ensure on the one hand a sufficiently fast convergence and on the other hand that the optimum is not missed (cf. Figure 2) [28, p. 437-478]. A good learning rate is often determined experimentally, but other approaches, as for example cyclical varying and cosine annealing, have been proposed [29, 6, 30].

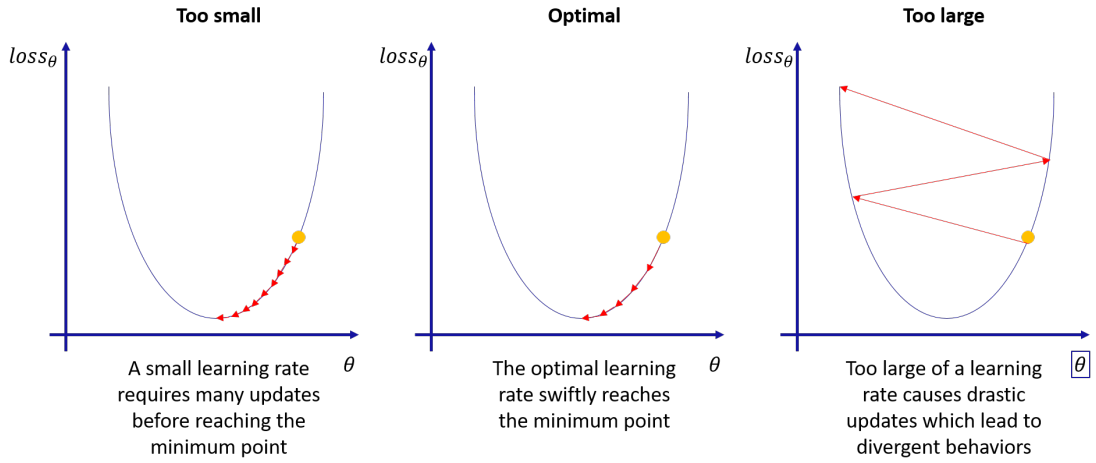


Figure 2.: Effects of different learning rates on a training process of a machine learning classifier (cf. [1])

Gradient descent does not guarantee to converge to a global optimum but it guarantees to converge at least to a local minimum where the gradient becomes 0 [8, p. 83-84].

Stochastic Gradient Descent

A gradient is calculated over all n training samples $\in D$. If n is large, determining the gradient is expensive, which is the reason why the more efficient variant *stochastic gradient descent* (SGD) is often used in practice [8, p. 290].

The stochastic gradient descent does not calculate the gradient on the entire set of training data. Instead, it randomly draws a subset called *batch* from the training data that is then used in the gradient calculation. Algorithm 1 shows SGD pseudocode. The learning rate η is often not constant but decreases over time, because the random sampling introduces noise which also affects the wanted minimum where the true gradient would normally become 0. Reducing the learning rate balances this noise [8, p. 291].

Generally, larger values for k result in more accurate approximation of the true gradient. The optimal batch size depends on the available hardware as it determines the trade-off between accuracy, throughput, and the degree of possible parallelization [8, p. 276].

Algorithm 1 Stochastic gradient descent (cf. [8, p. 291])

INPUT: Learning rate schedule η_1, η_2,...; initial θ_0

OUTPUT: final $\hat{\theta}$

$t \leftarrow 0$

while stopping criterion not met **do**

$B \leftarrow_{\$} D^k_{train}$ ▷ Sampling k elements from D_{train} uniformly at random.

$\hat{g} \leftarrow \frac{1}{k} \nabla \sum_{i=0}^{k} loss(f_{\theta_t}(x_i), y_i)$

$\theta_{t+1} \leftarrow \theta_t - \eta_t \hat{g}$

$t \leftarrow t + 1$

return $\hat{\theta} = \theta_t$

2.2.2. Neural Networks

Federated learning is the central topic of this thesis and describes a privacy-preserving process to train machine learning models. It will be introduced in detail in Section 2.3. Generally, the process can be adapted to any kind of machine learning technique, but many publications about federated learning focus on a technique that is called *neural network* or *deep learning* [6, 7, 4]. In the scope of this work we deal only with federated learning with neural networks. The terms denote a range of different algorithms, as for example convolutional neural networks, long short-term memory, auto-encoders, etc. They perform differently depending on data and applications. Convolutional neural networks are, for example, widely used in image classifications tasks, and long short-term memory is well-known for having a good performance in speech recognition [31, 32, 33, 34]. These algorithms have in common that they are inspired by the biological neuron system of the brain, and they typically belong to supervised learning [3, 2]. Figure 3 shows a representation of a biological neuron next to a neuron of an artifical neural network.

An artificial neuron takes — similar to a biological neuron — several input signals $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x} = (x_1, ..., x_i, ..., x_n)$ to produce an output signal $y \in \{0, 1\}^m$ or $y \in \mathbb{R}^m$. A graph consisting of connected neurons is called a neural network [26]. A neural network that contains only one neuron is a *perceptron* [35]. The first more advanced neural networks that appeared were *multi-layer perceptrons*. They were developed in many variants in the 1970's and 1980's together with a training mechanism that is called *backpropagation* [36, 37, 38]. Initially, they were of limited usability due to a lack of data, computationally limited resources, and the unavailability of an efficient learning algorithm for deeper networks [39, 36].

Neural networks usually contain many neurons grouped in layers. The first layer takes the input that shall be classified, and all other layers take the output of the previous layer as input, as Figure 4 shows. Layers between the input and the final output layer

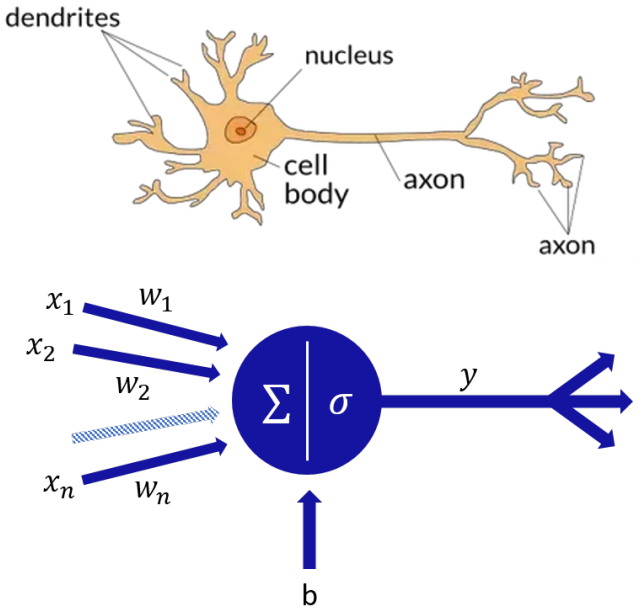


Figure 3.: Schematic representations of a biological and artificial neuron (cf. [2])

are called *hidden* layers. These additional layers allow to solve more complex problems, while a perceptron is only capable to learn linearly separable patterns. Layers do not have to be fully connected and can contain cycles. In the following, we consider neural networks that do not contain cycles, called *feedforward neural networks* [8, p.164 -165].
The output of a single neuron is derived as:

$$y = \sigma(\sum_{i=1}^{n} w_i x_i - b) \tag{8}$$

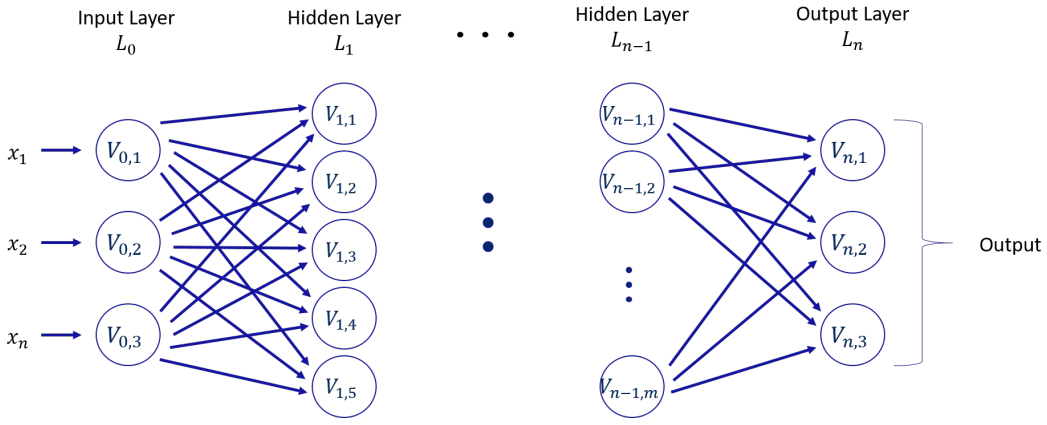


Figure 4.: Schematic representation of a feedforward neural network (cf. [3, p. 270])

x_i denotes the input data which is multiplied with the *weights* w_i. The *bias* b is added and the result is passed through a scalar function $\sigma : \mathbb{R} \to \mathbb{R}$ called the *activation function*. Typical activation functions are: the sigmoid function $\sigma(a) \to \frac{1}{1+e^{-a}}$, the threshold function $\sigma(a) = \mathbb{1}_{a>0}$, and the rectifier $\sigma(a) = max(0, a)$. The final output is denoted by y. Weights and biases are trainable variables that are optimized during the training process such that the final configuration $f_{\hat{\theta}}$ minimizes the *loss*, i.e., the difference between predictions of the network and the real labels [3, p. 269].

A feedforward neural network can be represented with a directed acyclic graph $G = (V, E)$ and a parameterized function $f_\theta : \mathbb{R}^N \to \mathbb{R}^M$. The function f_θ maps an input $x \in \mathbb{R}^N$ to an output $y \in \mathbb{R}^M$. Its output y is either directly the label of the class which the model predicts for the given input or a vector of probabilities for all possible classes such that x belongs to the class with the label $max_{i \in M}(y_i)$. θ represents the matrix of trainable parameters.

Finding a global minimum is hard, therefore approximative solutions, as for instance with SGD (cf. Section 2.2.1), are often used to find at least a local minimum in reasonable time.

Backpropagation

In the optimization process with SGD for neural networks, the gradients can be calculated with the *backpropagation algorithm* represented in Algorithm 2. In a first step, $f_\theta \leftarrow_\$ F_\Theta$ is randomly initialized and then evaluated with *feedforward propagation*. In this evaluation, an input x is passed through the neurons of all layers as in lines 3-5. The error between the output of the feedforward propagation and the true label is calculated in line 6 and "propagated backwards" from the last layer to the input layer. Based on the idea that errors at layer k are influenced by errors made in later layers $l > k$, the algorithm uses each gradient from a layer to calculate the gradient of the previous layer (cf. lines 7-8). The resulting gradients are then used in lines 10 and 11 to tune the weights and biases of θ such that the objective function *loss* is minimized [40, 3, p. 277-281].

Algorithm 2 Backpropagation (cf. [3, p. 278])

INPUT: $(\mathbf{x}, \mathbf{y}) \in D_{train}$, initial $\theta = \{W, b\}$, activation function σ

OUTPUT: best $\theta = \{W, b\}$

foreach $(\mathbf{x}, \mathbf{y}) \in D_{train}$ **do**
  $\mathbf{a}^{\mathbf{x},0} \leftarrow \mathbf{x}$ ▷ Set input activation
  foreach Layer $l \in \{1, \ldots, L-1\}$ **do** ▷ Forward pass
    $\mathbf{z}^{\mathbf{x},l} \leftarrow \mathbf{W}^l \mathbf{a}^{\mathbf{x},l-1} + \mathbf{b}^l$
    $\mathbf{a}^{\mathbf{x},l} \leftarrow \sigma(\mathbf{z}^{\mathbf{x},l})$
  $\boldsymbol{\delta}^{\mathbf{x},L} \leftarrow loss(\mathbf{a}^{\mathbf{x},L}, y)\sigma'(\mathbf{z}^{\mathbf{x},L})$
  foreach Layer $l \in \{L-1, L-2, \ldots, 2\}$ **do** ▷ Backpropagate error
    $\boldsymbol{\delta}^{\mathbf{x},l} \leftarrow ((\mathbf{W}^l)^T \boldsymbol{\delta}^{\mathbf{x},l+1})\sigma'(\mathbf{z}^{\mathbf{x},l+1})$
foreach $l \in L, L-1 \ldots 2$ **do** ▷ ▷ Gradient descent
  $\mathbf{W}^l \leftarrow \mathbf{W}^l - \frac{\eta}{|D_{train}|} \sum_{\mathbf{x}} \boldsymbol{\delta}^{\mathbf{x},l} (\mathbf{a}^{\mathbf{x},l-1})^T$
  $\mathbf{b}^l \leftarrow \mathbf{b}^l - \frac{\eta}{|D_{train}|} \sum_{\mathbf{x}} \boldsymbol{\delta}^{\mathbf{x},l}$

2.2.3. Dimension Reduction

For one of our defense layers, that we design to thwart backdooring attacks on federated learning and that we will introduce in Section 4, we use dimension reduction techniques to reduce the number of variables of the activations in the last hidden layer to avoid unwanted behavior in the later clustering [41, 42]. We aim to lose as less as possible information in terms of distinguishing features between the different samples, and test two different techniques for this purpose: Principal Component Analysis (PCA) and Fast-Independent Component Analysis (FastICA). We introduce them in the following two paragraphs.

Principal Component Analysis

PCA [43, 44] is a well-known algorithm to reduce dimensions. The basic idea is to transform the original coordinate system (the orthogonal normal basis) into a new one, such that the new one maximizes the variance of the data along its axes, called *principal components*. PCA can be done in four main steps. Each data sample of the original data has initially R coordinates. First, the data has to be standardized by subtracting the mean and dividing by the standard deviation. The next step is to calculate a covariance matrix of the R parameters. From the covariance matrix, all R eigenvectors with the corresponding eigenvalues are then retrieved (step 3). Eigenvectors are equal to the principal components that maximize the variance and eigenvalues correspond to the variance of its eigenvector. To reduce the data to $R - i$ dimensions while preserving

as much as possible information, the x eigenvectors with the lowest i eigenvalues are removed. The remaining eigenvectors are sorted in descending order of their eigenvalues into the columns of a matrix called *feature vector*. The transposed feature vector is then multiplied with the original data to receive the final data transformed towards the new orthogonal basis (step 4).

(Fast)-Independent Component Analysis

The second method for dimension reduction, *FastICA*, was published by Hyvärinen in [45] in 1999. It is an efficient technique to solve the Independent Component Analysis (ICA) problem that aims to linearly transform data represented in multidimensional vectors into maximally statistically independent components [46, 47, 48]. The idea is to divide a multivariate signal (i.e. outputs from several sources) into its sub-components.

ICA is not a dimension reduction technique itself, but allows to discover independent signals in data. It assumes a generative model for the observed data x that contains a linear combination of the independent variables (signals) $s_i, i \in \{1, ..., n\}$ with latent variables $a_j, j \in \{1, ..., n\}$:

$$x = As = \sum_{n}^{i=1} a_i s_i \tag{9}$$

The goal is to estimate the weight matrix W that transforms the data into the independent components: $s = Wx$. Before applying an ICA technique, the data is normally preprocessed in two steps to simplify the calculations. The first step is the centering, which involves subtracting the sample mean to get zero-centered variables. The second step, called whitening, transforms the data vector linearly to obtain a "white" vector whose components are uncorrelated and have a unity variance. Afterwards, the method maximizes the *nongaussianity* of the matrix (as metric for statistical independence) to recover the statistically independent variables. It is, intuitively, a rotation of the whitened matrix.

The basic FastICA has three main steps that are executed for every single component that shall be extracted: In the first step, it creates a random initialization for a weight vector w. In step 2, it calculates $w^+ = E\{xg(w^T X)\} - E\{g'(w^t x)\}w$. $E\{...\}$ denotes the average over all columns of the data x, $g()$ and $g'()$ are the first and second derivatives of a function $f()$ which approximates a measure for nongaussianity. Then, w is updated by determining the unit vector of step 2's output: $w = \frac{w^+}{|w^+|}$ (step 3). If the dot product of the old and new value for w is close to 1, which means that they have the same direction, the algorithm has converged and stops. Otherwise, it repeats from step 2 [48]. To determine the full weight matrix W, the process needs to be done for every additional component. Refer to [45] for more details.

In ICA, the number of components can be set to a smaller number to reduce the

dimensionality of the data.

2.2.4. Clustering

Clustering is an unsupervised machine learning technique that groups a set of objects into classes of similar objects. In a good clustering result, the distance between two clusters should be maximal, while the distances between elements of the same cluster should be minimal [49, pp. 1-2]. In the following, we introduce two different clustering approaches and the silhouette score that we apply in our second defense layer.

K-means

K-means is a well known partitional clustering technique. It iteratively divides data into k subsets.

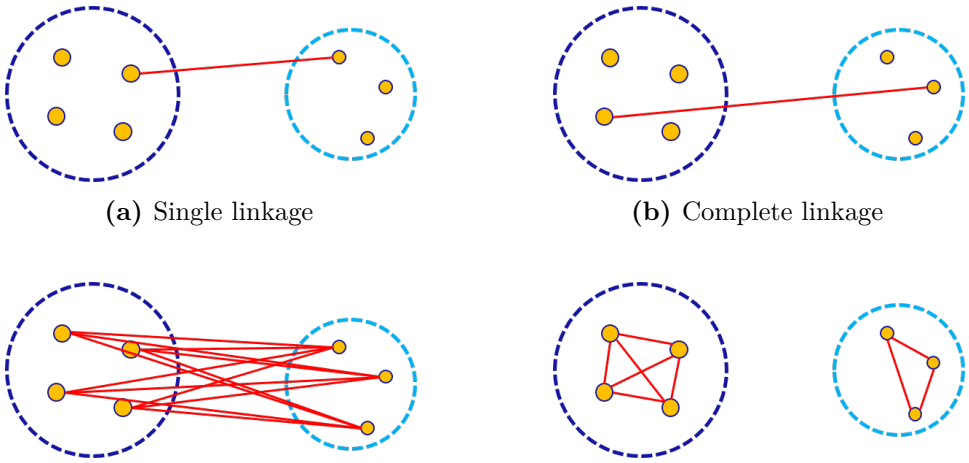
Each element in the data is defined by an R-dimensional point describing the element and is denoted by $E_i = (e_{i,1}, e_{i,2}, ..., e_{i,R})$. The algorithm starts with randomly creating cluster centroids C_1, C_2, ..., C_k which represent the centers of the k clusters. Each centroid is an R-dimensional point. In the next step, every data element is assigned to the cluster with the closest centroid. The distance between a centroid and a data element is determined with the euclidean distance. After the assignment, all centroids are updated by computing the arithmetic mean of the elements assigned to their cluster. The process of first assigning the data elements to the closest cluster and then updating the centroids is repeated until either the centroids do not change anymore, they almost do not change anymore, or after a predetermined number of iterations.

K-means can only detect convexly shaped clusters. R is equal to the number of dimensions we receive after dimension reduction.

Agglomerative Clustering

Agglomerative clustering denotes a type of hierarchical clustering that creates clusters by merging similar/close objects together. The merging stops when it reaches the number of requested clusters [50, pp. 71-72]. To decide which objects are similar/close different criteria can be applied. Popular criteria are ward, complete linkage, (group) average linkage, and single linkage. Figure 5 shows the four merging criteria.

- The *ward criterium* minimizes the variance in clusters by optimizing the pairwise squared difference between elements of one cluster [51]. This means the two clusters will be merged that increase the sum of pairwise squared difference the least.



(a) Single linkage **(b)** Complete linkage

(c) Average linkage **(d)** Ward criterium

Figure 5.: Visualization of four linkage criteria for agglomerative clustering

- *Complete linkage* calculates the pairwise distances between all elements, and searches for the maximum possible distance of two elements in different clusters. The two clusters with the minimum maximal distance will be joint.
- *Average linkage* merges the two groups of elements that have the minimum average distance when comparing all elements of one group with all elements of the other group.
- *Single linkage* joins the two clusters which contain the two closest elements that do not belong to the same cluster [50, pp. 72-78].

Silhouette Score

The *silhouette score* [52] is a measure that assesses how well a sample i fits into the cluster A it is assigned to compared to other clusters. It is calculated for a sample by determining its average distance to all other clusters. The average distance of a sample to a cluster thereby denotes the mean distance to every element in that cluster. Thus, for an element i the distance $a(i)$ to its own cluster A and the distance $b(i)$ to one other cluster B can be measured as follows:

$$a(i) = \frac{1}{|A| - 1} \sum_{i,j \in A, i \neq j} d(i,j) \tag{10}$$

$$b(i) = \min \frac{1}{|B|} \sum_{j \in B, i \neq B} d(i,j) \tag{11}$$

Based on these values, the silhouette score $s(i)$ for the element i is calculated in the

following way by comparing the own cluster A to the closest other cluster B:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{12}$$

$s(i)$ lies in a range from -1 to 1. A negative value indicates that the sample might have been assigned to the wrong cluster, a value around 0 says that the element is very close to another cluster, and a value close to 1 suggests that it fits well into the assigned cluster. To determine the silhouette value for a complete cluster, the silhouette values of all samples are averaged. This value is also called *overall average silhouette width*.

2.3. Federated Learning

2.3.1. Additional Notation for Federated Learning

We introduce further notation that is helpful for defining federated learning. Throughout the thesis, a model f_θ owned by a server at time t will be denoted with G^t. A model f_θ owned by a client i at time t will be represented with L_i^t.

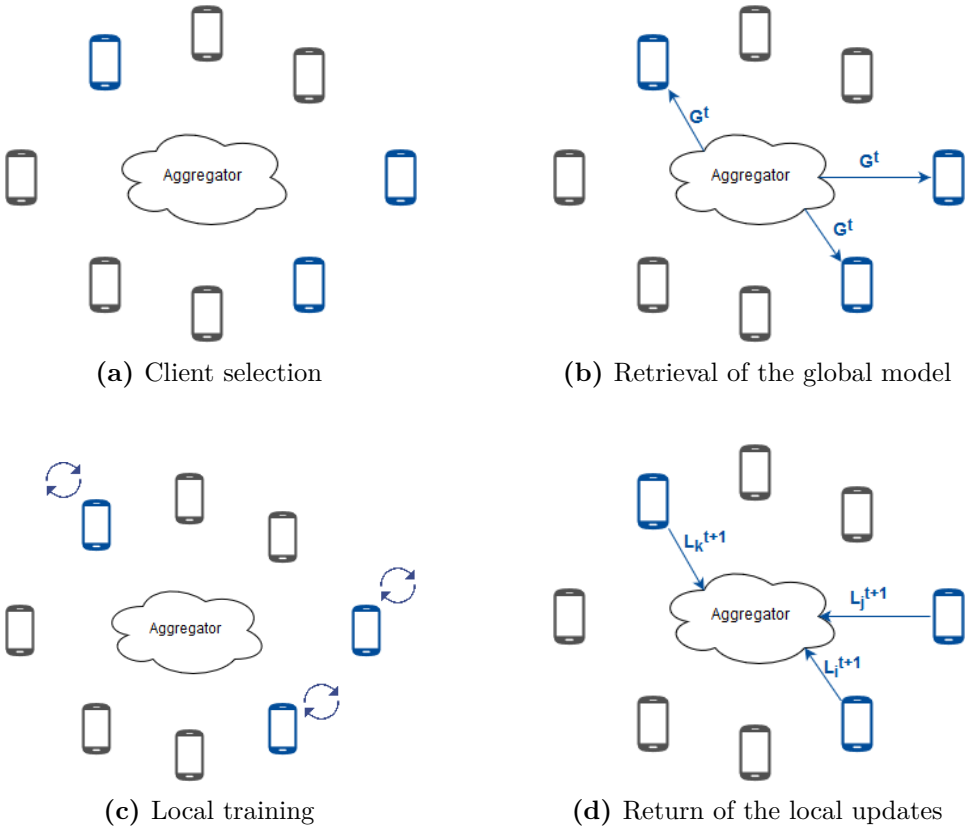
2.3.2. Federated Learning of a Machine Learning Classifier

Federated learning, introduced by McMahan et al. in 2017 [6], is a specific form of distributed learning. Distributed learning leverages data and/or computing power of several systems, for example data centers, to jointly train a machine learning model [53]. Federated learning was designed to train machine learning models with massively distributed data in a privacy-preserving manner [6].

The idea is that a central party — the aggregator — S learns a model for a prediction task from hundreds, thousands, or even millions of clients. It leverages the computing power available on the clients' devices to accelerate the learning process and to minimize the amount of required rounds of communication necessary to achieve a sufficiently accurate model.

In every iteration, the aggregator shares its current global model G_t with a randomly selected fraction C of the available set of clients. Each selected client k performs the training with SGD with the global model G_t and returns the resulting average gradient $g_k = \nabla loss(f_{\theta_t})$. Figure 6 shows an overview of the federated learning process.

After receiving all updates, the aggregator averages them and adds the result to the global model. This baseline operation for combining the updates — called $FederatedSGD$ — is represented in Equation 13. K denotes the number of clients selected per round (so $K = C \cdot \#\text{clients}$), η is the global learning rate, n_k the amount of local training data of client k, and $n = \sum_{i=0}^{K} n_i$ indicates the total number of training



(a) Client selection **(b)** Retrieval of the global model

(c) Local training **(d)** Return of the local updates

Figure 6.: Overview of the federated learning process

samples [6].

$$G^{t+1} \leftarrow G^t + \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k \tag{13}$$

After updating the global model, the aggregator can decide if it wants to repeat the process. For instance, when the aggregator has access to a test data set, training with federated learning can be stopped when no significant performance improvement is measured for a specific time frame. Throughout the complete process of federated learning sensitive data of clients never needs to be shared and remains locally on the device, thus improving the privacy protection of the training data.

Federated learning has four main characteristics which distinguish it from normal distributed learning that is often used in data centers:

- **Not Independent and Identically Distributed (Non-IID)**: The local data set of a client is normally not representative for the distribution of the population, and different local sets are likely drawn from different distributions. In contrast, data from devices that belong to the same user can be dependent. This can cause

a high variability in the submitted local updates that is balanced by the averaging process. Instead, in distributed learning independent and identically distributed data is normally assumed [6, 54, 55, 56].

- **Unbalanced**: The amount of local training data available at each client can vary significantly which is in contrast to distributed learning — e.g., among data centers — where data splitting can be done in a balanced manner such that every node has (nearly) the same amount of data per class [6, 56, 54].
- **Distributed**: The number of clients is much larger than the average available amount of training data on every device. Additionally, while in distributed learning there may be hundreds or thousands of nodes, in federated learning millions of users should be able to contribute [6].
- **Limited Communication**: The amount of necessary communication should be minimal, because mobile devices tend to have limited bandwidth and to go frequently offline. In distributed learning — e.g., among data centers — a stable connection with sufficient throughput is the norm [6].

2.3.3. Federated Averaging

Instead of using the baseline algorithm, McMahan et al. propose $FederatedAveraging$ to increase the local computation per round per client, so that the number of iterations and therefore the communication costs are reduced. In $FederatedAveraging$, the clients do not return the resulting gradients. Instead, each client k updates its local model to L_k^{t+1} and returns the difference between the global model and its new local model: $G^t - L_k^{t+1}$. The aggregator updates the global model by averaging the local updates, as Algorithm 3 shows [6].
In this process, there are several parameters that can be adjusted to regulate the communication and computation costs as well as the generalization capabilities of the model. C is the proportion of clients that are selected in every iteration for training on the local data, B is the size of the local batches in which each client splits its data set during the local training, and E is the number of epochs a client executes in each local training with its data.

In [6], McMahan et al. state that increasing the proportion of clients C reduces the number of iterations needed to reach convergence, but depending on the batch size B the effect varies. For larger batch sizes it is less significant than for smaller ones. They suggest to fix C to 0.1 for a good balance between efficiency and convergence. Furthermore, they use a global learning rate of $\eta = 1$. Bagdasaryan et al. [7] point out that if the global learning rate η is set to $\frac{n}{K}$, the previous global model will be completely replaced by the averaged updates.

Algorithm 3 Federated averaging (cf. [6])

Server executes:
  initialize w_0
  foreach round $t = 1, 2, \ldots$ **do**
    $m \leftarrow \max(K, 1)$
    $S_t \leftarrow$random.choice($clients, m$) ▷ Select m random clients
    foreach client $k \in S_t$ in parallel **do**
      $L_k^{t+1} \leftarrow$ ClientUpdate(k, G_t)
    $G^{t+1} \leftarrow G^t + \eta \sum_{k=1}^{K} \frac{n_k}{n} L_k^{t+1}$

ClientUpdate k,G^t: ▷ Run on client k
  $L^{t+1} \leftarrow G^t$
  $\mathcal{B} \leftarrow$ (split P_k into batches of size B)
  foreach local epoch i from 1 to E **do**
    foreach batch $b \in \mathcal{B}$ **do**
      $L^{t+1} \leftarrow L^{t+1} - \gamma \nabla l(L^{t+1}; b)$ ▷ Gradient Descent to update model
  return $(G_t - L^{t+1})$ to server

2.3.4. Secure Aggregation

Although individual data remains locally at the client's device, the submitted update can still leak information through, for example, *model inversion-*, *user likability-*, or *membership inference-attacks* [57, 58, 59].

To approach this problem, Bonawitz et al. propose a secure aggregation protocol that calculates the weighted average of the clients' updates such that the aggregator is not able to access individual updates anymore [4]. Therefore, they tolerate drop-out of clients in a server-controlled and unauthenticated network. The authors propose two designs: one more efficient for the honest-but-curious security model, and a second that is also secure under the active-adversary model, but adds an additional communication round. Building blocks are for example Shamir's t-out-of-n Secret Sharing [60], the Diffe-Hellman key agreement [61], a (symmetric) authenticated encryption which satisfies *indistinguishability under chosen plaintext attack (IND-CPA)* and *integrity of ciphertext (INT-CTXT)*, a secure pseudorandom number generator, and an *universal forgery under chosen-message attack (UF-CMA)* secure signature scheme. Figure 7 gives an overview of the protocol's design. It consists of four rounds under the honest-but-curious security model and of five rounds in the active-adversary model.

In the 1st round, each participating client u creates two Diffie-Hellman public/private-key pairs $\langle c_u^{SK}, c_u^{PK} \rangle$ and $\langle s_u^{SK}, s_u^{PK} \rangle$. The two public keys c_u^{PK} and s_u^{PK} are sent to the server that broadcasts them to all other clients. In the active-adversary model, a Public-Key-Infrastruktur (PKI) is established before the 1st round, where all clients

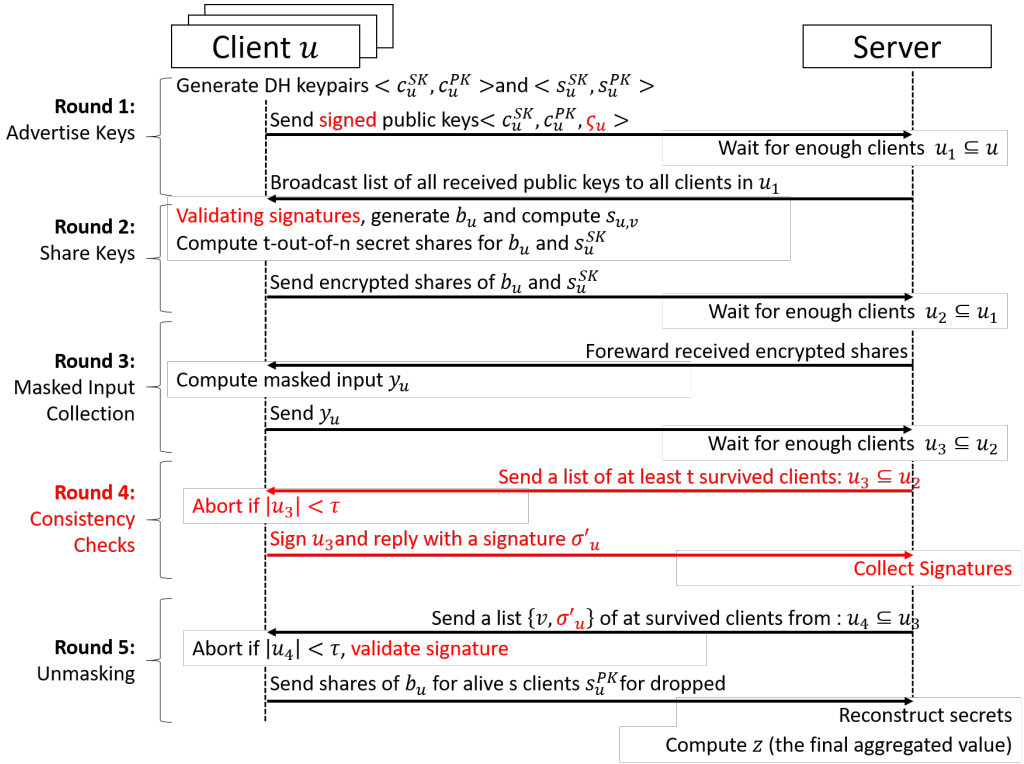


Figure 7.: Overview of the secure aggregation protocol by Bonawitz et al. [4]

register to create verifiable keys used ς_u to create signatures for their identities. It ensures that the server cannot simulate fake clients to break the privacy of honest parties. In this case, the public keys are signed with u's signature ς_u before they are sent to the server.

In the 2nd round, the signatures are validated in the active-adversary case. In the honest-but-curious setting, this step is skipped. Instead, u directly generates a random seed b_u and engages in a Diffie-Hellman key agreement with each other client. The resulting key of such an agreement between u and another client v is denoted by $s_{u,v}$. Client u then creates n shares following Shamir's Secret Sharing with a threshold τ for both b_u and the secret key s_u^{SK}, encrypts the share with the public keys of the other clients, and sends them to the server. In this process, n denotes the number of participating clients.

In the 3rd round, the server forwards all shares to the respective recipients. With this forwarding, they learn which other participants are still active. A client u uses the shared key $s_{u,v}$ with another client v as input for a PRG to calculate a random vector

$p_{u,v}$:

$$p_{u,v} = \Delta_{u,v} \cdot \mathbf{PRG}(s_{u,v}), \text{ where } \Delta_{u,v} = 1 \text{ when } u < v \text{ and } \Delta_{u,v} = -1 \text{ when } u > v \tag{14}$$

This is done for all other active clients. In the next step, u calculates a second random vector $p_u = PRG(b_u)$. The random vectors are used to hide the secret vector x_u. The result y_u is sent to the server:

$$y_u \leftarrow x_u + p_u + \sum_{\forall v \in Active} p_{u,v} \tag{15}$$

The 4th round is only required in the active-adversary model. In this round all active clients receive a list that contains all clients that were still active at the end of round 3. This list is signed by each client u with its private key ς_u and returned to the server.

In the last round, the server again sends a list of all active clients at the end of the previous round (3 or 4). In the active-adversary model the active clients also receive the signatures that were created by the other active clients in round 4. This allows them to verify that all active clients received the same list, meaning that the server cannot lie about who has dropped out. If the number of active clients is less than τ, the aggregation stops. Otherwise, u sends the share $b_{v,u}$ for each client v who is still active in this round (denoted by set $\mathcal{U}_{active}$) and the share $s^{SK}_{u,v}$ for each client v that dropped out after round 3 (denoted by set $\mathcal{U}_3$). Afterwards, the server can reconstruct the private key s^{SK}_v for each client $v \in \mathcal{U}_3$ to recompute $p_{u,v}$ that was originally created with a still active client $u \in \mathcal{U}_{active}$. The random vectors b_u for all still active clients $u \in \mathcal{U}_{active}$ can also be recreated to determine p_u with the PRG. The final aggregation result of a communication round is then evaluated by the server S:

$$z = \sum_{u \in \mathcal{U}_{active}} x_u \tag{16}$$

$$= \sum_{u \in \mathcal{U}_{active}} y_u - \sum_{u \in \mathcal{U}_{active}} p_u + \sum_{u \in \mathcal{U}_{active}, v \in \mathcal{U}_3 \setminus \mathcal{U}_{active}} p_{v,u} \tag{17}$$

2.4. Data Poisoning

While federated learning protects the privacy of the clients, it also faces the risk that one or more of the clients can be malicious. “Malicious” can have different meanings here. Generally, we can group the objectives of adversaries into two groups [25]:

- **Untargeted Misclassification**: The adversary aims to decrease the overall model accuracy (and to increase the loss), or to prevent convergence [62, 63, 64]. Example: An untargeted misclassification occurs when the attacker causes all

samples of 7's in the MNIST data set [65] to be classified as any digit other than 7. (MNIST contains images of handwritten digits from 0 to 9.)

- **Targeted Misclassification**: The adversary wants to cause the model to misclassify certain samples of the population. This can either be one (or even more) complete class(es) or some specific samples.
  Example: A targeted misclassification occurs when the attacker aims to achieve that all 7's in the MNIST data set are classified as 1.

In this work we focus on the second type of objective. In Chapter 4 we formally define the problem and present more details.

3. Related Work

3.1. Attacks

The area of federated learning has not been studied from the security perspective in depth because of being relatively new. Nevertheless, the first attacks by malicious participants have been published. The distributed nature of federated learning, along with the infeasibility to vet participants at scale, leaves the door wide open for malicious clients who cannot only compromise the integrity of the model, but also can violate the privacy of other users. Moreover, mitigating the privacy threat through secure aggregation complicates the detection of malicious contributions.

In [7], Bagdasaryan et al. present a semantic backdoor attack which can be introduced into the global model by any malicious client who is selected by the server in just one round. They amplify a malicious update to cause the model replacement by scaling it such that it has more effect on the global model than honestly created updates. Additionally, they propose an objective function which includes the loss function chosen by the aggregator, but also accounts for anomaly detection for malicious clients. Furthermore, they suggest to clip weights to a maximum bound to evade detection. As their ideas rely on model replacement, the backdoor will only be persistent if it is inserted shortly before the training process converges. In earlier rounds, the injection of a malicious client is overwritten by the contribution of honest parties in just a few rounds, but in later rounds the model is close to convergence and the updates of the honest clients cancel each other out. Therefore, the model replacement is successful. Another aspect of their attack that enables to evade anomaly detection is that it is possible to split the necessary updates to achieve the backdoor between malicious clients. Nevertheless, as a client cannot control the random selection process the success of the attack depends on a malicious client being chosen late in the training process. The authors used the CIFAR-10 [66] and a Reddit data set in the experiments. They chose a subset of images from CIFAR-10 that contain common characteristics (e.g. green cars) for the malicious clients to create backdoors. In their setup, the honest clients do not have correctly classified data with the chosen characteristics. As the attacks proposed by Bagdasaryan et al. have been investigated in the scope of this work, a detailed overview of the parameters they use can be found in Table 2 next to the parameters tested by McMahan

et al. in [6] who initially proposed federated learning as a privacy-preserving approach for machine learning in Table 1.

In [25], Bhagoji et al. propose similar model poisoning attacks to create a backdoor for one single attacker-chosen example and they demonstrate how to evade detection via accuracy checking, the analysis of weight updates distribution, and L_p distances. In contrast to Bagdasaryan et al., they consider a single adversary that controls one client. To weaken the adversary, they assume independent and identically distributed data at the clients such that the aggregating server can detect malicious contributions easier than in the typical unbalanced and non-IID setting of federated learning. Similarly as Bagdasaryan et al., they also propose to combine the malicious samples with normal training data and jointly optimize both to reduce the probability of detection. Their findings suggest that detection via accuracy checking and weight analysis would allow to easily detect the malicious contribution while the backdoor accuracy remains low. Therefore, they propose an improvement of the attack where a malicious client alternately optimizes on its target samples and normal training data in local epochs. Thereby, only the adversarial part is boosted/scaled. Additionally, they suggest the usage of a constraint for weight updates with respect to the benign update to ensure that a local optimum that looks as benign as possible is chosen. This work requires either to assume that the contribution of the honest clients cancel each other out, or that the malicious client can estimate the aggregated contribution of the other honest clients chosen in a round. Fashion-MNIST [67] and Adult Census[1] are used to experimentally evaluate the proposed techniques.

Liu et al. [68] published one of the first works that designs trojaned backdoor attacks for neural networks. They start with a fully trained model which they poison. Namely, they reverse engineer one training sample for every class. They then choose one or more neurons in a hidden layer which are well connected and generate a trigger that maximally activates these neurons. The form of the trigger can be chosen by the attacker, but how the trigger is embellished in terms of colors is optimized such that it has a maximal effect on the activation of the chosen neurons. Afterwards, they retrain all layers after the layer that contains the chosen neurons to activate the backdoor. In this process, they use the reverse engineered data and the optimized trigger.

BadNets is an attack proposed by Gu et al. in [71] that aims to infiltrate an outsourced training process with trojaned backdoors. In this attack, a client c sends the model structure and the training data D_{train} to a server S which returns the trained parameters $\hat{\theta}$. In an adversarial setting, the server instead returns parameters of a backdoored model $\theta^* \neq \hat{\theta}$ which should preserve a good performance on the test data set D_{test} of the client (that the server cannot access) and outputs attacker-chosen predictions on

[1] `https://archive.ics.uci.edu/ml/datasets/adult`

inputs that contain the backdoor trigger: a specific pixel pattern. The authors evaluate the misclassification of backdoored images from one class i to another j with all possible combinations of i and j, as well as the simultaneous misclassification of all classes i to $i + 1$. Trojan backdoors require an active intervention of the attacker at both training and inference time because a pattern needs to be added. In *BadNets*, the authors add pixel-patterns on images from the MNIST data set [65] and a data set with US traffic signs.

Tan et al. [69] introduce an adaptive adversarial learning for neural networks that minimizes the differences between the distributions of activations extracted from the last hidden layer of benign and backdoor samples to evade defense mechanisms. They test how their attack, that injects trojaned backdoors, circumvents two defenses, namely *NeuralCleanse* by Wang et al. [80] and activation clustering by Chen et al. [77]. Their test setting assumes that the attacker has access to the model to manipulate it by retraining with their adapted objective function that takes the differences between the distributions into account. They use CIFAR-10 [66] and the GTSRB [70] data set.

There is no publication yet that focuses on trojaned backdoor attacks in the federated learning setting which is the reason why we include the works of Gu et al., Liu et al., and Tan et al. in this section. Bagdasaryan et al. [7] include just one experiment in which they show that this type of backdoor can also be injected with their attack scheme, because they consider it as weaker as semantic backdoors due to the data manipulation that is required to create and use such backdoors.

3.2. Defenses

Fung et al. [72] propose *FoolsGold* to mitigate sybils in federated learning poisoning by observing contribution similarity. The underlying assumption is that an adversary will try to use several infiltrated clients to cause a backdoor functionality. The authors use the cosine similarity of gradient updates to separate sybils from honest clients. The idea is that because malicious clients have the same goal their updates will exhibit similarities. In their experimental setting the authors use MNIST [65], KDDCup [73], and an Amazon data set [73]. Although the technique makes the model more resistant to attackers, it comes at the cost of privacy, as a history of a client's updates is required to compare similarities. They also explicitly state that their scheme is not successful in deep learning because of the non-convexity of deep neural networks that makes cosine similarity ineffective to detect colluding sybils. The authors also admit that their defense cannot detect a single malicious client and that it can be evaded if the attacker decomposes the target model into orthogonal update vectors and each colluding client contributes one of these vectors.

	Google [6]				
	MNIST	**CIFAR-10**	**Shakespeare**	**Social-Network Posts**	**Recommended**
Data type	Image	Image	Text	Text	-
Number of classes	10	10			-
NN-type	MLP & CNN	CNN	LSTM (866,578 parameters)	LSTM (4,950,544 parameters)	-
Data set size	training: 60,000; testing: 10,000	training: 50,000; testing: 10,000	training: 3,564,579; testing: 870,014	training: 10,000,000; testing: 100,000	-
Distribution characteristics	non-IID & IID, balanced	IID	non-IID, unbalanced; IID, balanced	non-IID, unbalanced	non-IID, unbalanced
Splitting technique	IID: shuffling, 600 random samples/client non-IID: 2 shards with 300 samples of 2 different classes/client	IID: 500 random training samples/client; 100 random testing samples/client	according to play & role	by author, at most 5,000 words, training uses different authors than testing	'most natural': Shakespeare splitting
Number of clients	100	100	1146	500,000	-
C (client fraction)	0.1, 0.2, 0.5, 1.0	0.0, 0.1, 1.0	0.1	$\frac{1}{2,500}$	0.1
E (local epochs)	1, 5, 10, 20, 25, 50, 100, 200, 400	1, 5, 10, 20	1, 5, 25, 50, 100, 200	1, 5	not given
B (local batch size)	10, 50, ∞	50, 100	10	8	not given
η (global learning rate)	1	1	1	1	not considered
γ (local learning rate)	IID: 0.215 non-IID: 0.1	FedSGD: 0.15, 0.45, 0.6, 0.7; decay: 0.9934/round FedAvg: 0.05, 0.15, 0.17, 0.3, 0.5; decay: 0.99/round	1.47	FedSGD: 6.0, 9.0, 18.0, 22.0; FedAvg: 3.0, 6.0, 9.0	experimentally with each data set
Main task accuracy	2NN: 97 %; CNN: 99 %	80 %, 82, %, 85 %	54 %	10.5 %	-
Number of rounds	2NN: 70-4,300; CNN: 16-1,200	3,000	41-4,000	1,000	-

Table 1.: Configurations by Google [6] for federated learning experiments

	Bagdasaryan et al. [7]	
	CIFAR-10	**Reddit**
Data type	Image	Text
Number of classes	10	50,000
NN-types	ResNet18 (2,700,000 parameters)	2-layer LSTM (10,000,000 parameters)
Data set size	training: 50,000; testing: 10,000	training: not explicitly stated testing: 5,034 posts
Distribution characteristics	unbalanced, not really non-IID	unbalanced, non-IID
Splitting technique	all classes per client, splitting according to Dirichlet (hyperparameter 0.9)	according to user (at least 150, at most 500 posts)
Number of clients	100	83,293
C (client fraction)	0.1	$\frac{100}{83,293}$
E (local epochs)	2	2
B (local batch size)	64	20
η (global learning rate)	1	800
γ (local learning rate)	0.1; decay factor: 10	20
Main task accuracy	92 %	19 %
Number of rounds	10,000	5,000

Table 2.: Configurations by Bagdasaryan et al. [7] for federated learning experiments

Byzantine-tolerant distributed learning can also be considered as a possible defense against model poisoning in federated learning. Blanchard et al. [62] propose *Krum* to tolerate f out of n byzantine clients while proving convergence of the SGD with the *Krum* function, but their data is independent and identically distributed. *Krum* calculates the sum of the closest $n - f - 2$ other local updates for each update and chooses the update with the minimum sum as global update instead of averaging all local updates. As slightly adapted version — *MultiKrum* — averages between the updates with the m minimum sums. However, Bagdasaryan et al. [7] state that using *Krum* strengthens their attack instead of defending against it, and that it also reduces privacy protection as single updates can be accessed and can leak information about the used training data.

SLSGD by Xie et al. [74] bases on a similar idea as *Krum*. The authors propose to use a trimmed mean instead of the standard averaging of federated learning to exclude malicious inputs. Afterwards, they apply a moving average on the trimmed mean to mitigate extra variance. The authors do not evaluate their defense on the backdooring-attacks presented in section 3.1, but only investigate the effectiveness against label-flipping attacks that aim to misclassify all samples of a class. This type of attack can already be detected with classical main task accuracy checking as a complete

misclassified class reduces it significantly. CIFAR-10 [66] is used in the experiments with label-flipping attacks.

Shen et al. published *AUROR* [75] which is a defense against label-flipping in distributed learning. The idea is to identify features that indicate malicious behavior by clustering the contributions of all participants into two sets for each feature and determining if the distance between the two sets is above a threshold α. Afterwards, the clustering results for all indicative features are analyzed. Users that appear more than τ times in a cluster with less than $\frac{n}{2}$ elements are marked as malicious and their contributions are removed from the update. In the experiments, MNIST [65] and the GTSRB [70] data sets are used and the features are gradients. This approach assumes independent and identically distributed data and Bagdasaryan et al. [7] claim that in a non-IID setting honest updates would be removed such that the main task accuracy would be affected. Additionally, they state that their clipping approach evades detection by *AUROR*. Shen et al. also mention that mixing benign and malicious training data decreases the probability of detection of a malicious client through such a method, but it also harms the attack's success.

STRIP was recently proposed by Gao et al. [76] for detecting trojaned inputs at inference time. The authors suggest to apply perturbation on incoming classification requests and to test how the perturbation changes the classification output. If there is not enough variance in the outputs, *STRIP* indicates that the input was trojaned. *STRIP* can only detect if a trained model was already poisoned, but it cannot defend in real-time against poisoning or recover from poisoning. *STRIP* is designed and tested only for image data sets. They evaluate their approach with MNIST [65] and CIFAR-10 [66].

Chen et al. [77] published another work that investigates defenses against trojan backdoors. They use activation clustering to detect poisoned classes. After creating a model with potentially poisoned data, the activations of the last layer of the training data are extracted. They are grouped according to the predicted class, and each group is clustered with 2-means. The authors suggest that a poisoned class results into two classes, while unpoisoned classes can be well represented with one cluster. Additionally, they suggest the summary/average of a class as verification for poisoning because it will show poisoned elements. If poisoning is detected, they propose to recover by training with the correctly relabeled, previously poisoned data until convergence. The experiments in this work are done with MNIST [65], LISA traffic signs [78], and the Rotten Tomatoes movie review data set [79].

Neural Cleanse by Wang et al. [80] aims to detect and patch trojaned backdoors at inference time by comparing the distances between classes. The authors determine the minimum modification of data from all possible classes necessary to misclassify them to

a target class t with a given trained model f_θ. If the necessary modification is smaller than a threshold this class is marked as poisoned with a trojan backdoor. The necessary modification can be considered as the reversed trigger and it can be used to patch the poisoned model. The authors propose to patch by pruning neurons that cause the most significant differences between clean and poisoned data or by using the reversed trigger for re-training. Alternatively, adversarial inputs can also be refused by the model after detecting the backdoor without patching the model. Neural cleanse is only designed for image data. It assumes that a clean set of data is available to determine distances between the classes. The authors use MNIST [65], GTSRB [70], YouTube Face [81], and PubFig [82] for their tests and rely on attacks from *BadNets* [71] and Liu et al. [68]. As pointed out by the authors *Neural Cleanse* can be (partially) circumvented by large triggers and multiple infected classes with different triggers.

Liu et al. propose a similar idea in [83]. In their work, they consider a "Machine learning as a service"-scenario where a client outsources training data and model specifications to a service provider which trains a model based on the given information. After returning the fully trained model, the client can verify its quality with a validation data set unknown to the server. But a malicious service provider can inject trojaned backdoors without harming the validation data set accuracy because of the sparsity of deep neural networks. Liu et al. suggest a *fine-pruning* defense: the client first analyzes the activations of samples from its validation data set based on the returned model, and then removes neurons that have a low average activation until the accuracy drops too significantly. Afterwards, it retrains the remaining network with a small set of clean data to be able to also defend against optimized attacks that take the pruning into account.

In [84], McMahan et al. introduce *user-level differential privacy* for federated learning which provides privacy concerning the presence or absence of one particular user in the training data. It is achieved by four modifications of the original federated learning process. First, in every iteration they do not choose a fixed proportion of clients, but a random number of clients. Thus, the number of clients for an update changes every round. Second, they clip the L_2 norm of local updates and they propose two possible estimators for combining the local updates. Both together allow to estimate the sensitivity of an update such that an appropriate level of noise can be added. Finally, after the aggregation, gaussian noise is added to the new global model. While user-level differential privacy is — as the name says — meant to protect privacy, Bagdasaryan et al. [7] point out that it can also be used for defending poisoning attacks. They test their attack with the clipping mechanism and adding noise on the global model and show that differential privacy impedes but does not evade the attacks. Additionally, adding noise also reduces the empirical accuracy of a model. The mechanism proposed

by McMahan et al. is not applicable to all kinds of federated learning applications as for example image classifications. A similar idea to protect privacy on client level with differential privacy is also published by Geyer et al. in [85]. Both works base upon the same ideas and mainly differ in details in the experimental setup as for example that McMahan et al. use the Reddit data set and Geyer et al. use MNIST [65].

4. Approach

In the following chapter, we present our proposal to defend federated learning against a specific type of model poisoning attacks, namely semantic backdoors. We provide the first formal definitions for the three different types of backdoor attacks, before limiting the scope to the attack type analyzed in this thesis. Afterwards, we introduce the threat model, and the requirements for a defense system, before presenting the details of the three defense layers.

4.1. Problem Definition

Intuitively, backdooring a machine learning model means that an attacker adds a secret functionality to it. While it performs naturally on the majority of possible input samples, a small, attacker-chosen set exists which will not be predicted to its true label but to an attacker-chosen target class instead. In federated learning, such a backdoor can be injected into a model via a malicious local update. Therefore, this type of poisoning is called *model poisoning* instead of *data poisoning* which is known from older machine learning techniques, and entails manipulating training data used to create a poisoned model.

Although we focus on semantic backdoors, which is a specific type of backdooring attacks, in the context of federated learning in this work, for the sake of clarity, we first define the concept of a "backdoored classifier" and its different types in general terms, regardless of whether the training process is centralized, distributed, or federated.

Concretely, we define the adversarial objective against a classifier f_θ in terms of malicious behavior that f_θ should exhibit in order to be declared "successfully backdoored", independently of how the attacker operates. We will then lift this concept to the federated learning scenario, specifying realistic adversarial capabilities, and complementing the adversarial objectives accordingly.

Firstly, we introduce some notation to formally define the problem. For a set of instances $X \subseteq \mathcal{X}$ and a label $y \in Y$, we denote by $X_y := \{x \in X : f(x) = y\}$ the set of samples in X that belong to class y, and refer to those samples as the y-instances in X. Given a classifier f_θ, for any two (possibly coinciding) labels $y, y' \in Y$ we denote by $X_{y \to y'} := \{x \in X_y : f_\theta(x) = y'\}$ the set of y-instances in X that f_θ predicts as y'.

To assess the success of a backdooring attack, a metric called *backdoor accuracy* is used. It measures the proportion of the samples in an attacker-chosen set X^* that are misclassified to the target label y_t.

Definition 1. *Let $X \subseteq \mathcal{X}$ be a set of instances, $y \in \mathcal{Y}$, and $X^* \subset X_y$[1]. We define the 'backdoor accuracy' of f_θ^* w.r.t. to a set X^* and target y_t as the number of samples in X^* that f_θ^* misclassifies as belonging to the target class:*

$$acc^{bd}_{X^*,y_t}(f_\theta^*) := \frac{|\{x \in X^* : f_\theta^*(x) = y_t\}|}{|X^*|} = \frac{|X^*_{y \to y_t}|}{|X^*|} \tag{18}$$

To introduce a backdoor into a machine learning model, we assume that an attacker A can control up to $m \ll K$ clients. The attacker's goal is to persistently manipulate the global model G^t such that it outputs an attacker-chosen classification on attacker-chosen input which is called a *misclassification functionality* or *backdooring functionality*. Through such an injection, the main task accuracy should not be affected, as this would be easy to detect and possible anomaly-detection mechanisms should be evaded. The attacker regulates the contributions of the controlled clients such that they update the global model into the desired direction.

Definition 2. *A classifier f_θ^* for a problem $f : X \to Y$ exhibits a 'targeted misclassification functionality' w.r.t the test data set D_{train} and in comparison to the clean model f_θ if the following conditions hold:*

1. *f_θ^* performs well on 'natural' inputs:*

$$acc_{D_{test}}(f_\theta^*) \approx acc_{D_{test}}(f_\theta) \tag{19}$$

2. *f_θ^* exhibits malicious behavior on 'malicious' inputs, i.e. it has a good 'backdoor accuracy':*

$$acc^{bd}_{D_{adversary},y_t}(f_\theta^*) = \frac{|D_{adversary_{y \to y_t}}|}{|D_{adversary}|} \approx 1 \tag{20}$$

As described in Chapter 3, in [72] the authors propose a data poisoning technique that aims at misclassifying a complete class which is therefore difficult to accomplish without reducing the empirical accuracy of the model. This poisoning attack is called *label-flipping attack*.

Definition 3. *A classifier f_θ^* for a problem $f : X \to Y$ exhibits a 'label-flipping functionality' if the following conditions hold:*

[1] A more general concept of "backdoored set" X^* that is not restricted to one single class y would also make sense. We opted for the specific, one-class version as it captures all attacks we consider (and simplifies the notation compared to the general version).

1. *f_θ^* performs as the not-poisoned model f_θ on all data that does not belong to the target class C_{A_t}:*

$$acc_{D_{test} \setminus f^{-1}(\{C_{A_t}\})}(f_\theta^*) \approx acc_{D_{test} \setminus f^{-1}(\{C_{A_t}\})}(f_\theta) \tag{21}$$

2. *f_θ^* causes a misclassifcation of all elements of an attack specified class C_{A_t} with its label y_t:*

$$\forall x \in f^{-1}(\{C_{A_t}\}), f_\theta^*(x) = y_t \neq f(x) \tag{22}$$

In a backdoor(ing) attack, just specific characteristics of the training data cause the targeted misclassification. The malicious inputs can appear naturally in the data (*semantic backdoor*). The work by Bagdasaryan et al. [7], that is described in Chapter 3, focuses on semantic backdoor attacks. The authors aim at misclassifying green cars as birds while all other cars are correctly classified. Alternatively, samples for backdoor attacks can be specially crafted with pixel-patterns or other artificially added properties (*trojan backdoor*) [71, 76, 76].

Definition 4. *A classifier f_θ^* for a problem $f : X \to Y$ exhibits a 'semantic backdoor functionality' if the following conditions hold:*

1. *f_θ^* performs as the not-poisoned model f_θ on 'natural' inputs:*

$$acc_{D_{test}}(f_\theta^*) \approx acc_{D_{test}}(f_\theta) \tag{23}$$

2. *f_θ^* causes a misclassification of all elements of attacker specified data $D_{adversary}$ to a label y_t which is not equal to the true label $f(x) = y$:*

$$acc^{bd}_{D_{adversary}, y_t}(f_\theta^*) = \frac{|D_{adversary_{y \to y_t}}|}{|D_{adversary}|} \approx 1, \forall (x, y) \in D_{adversary}, f_\theta^* = y_t \neq f(x) \tag{24}$$

3. *$D_{adversary}$ appears in the natural data:*

$$D_{adversary} \subseteq (D_{train} \cup D_{test}) \tag{25}$$

Definition 5. *A classifier f_θ^* for a problem $f : X \to Y$ exhibits a 'trojaned backdoor functionality' if the following conditions hold:*

1. *f_θ^* performs as the not-poisoned model f_θ on 'natural' inputs:*

$$acc_{D_{test}}(f_\theta^*) \approx acc_{D_{test}}(f_\theta) \tag{26}$$

2. *f_θ^* causes a misclassifcation of all elements of attacker specified data $D_{adversary}$ to a label y_t which is not equal to the true label $f(x) = y$:*

$$acc^{bd}_{D_{adversary},y_t}(f_\theta^*) = \frac{|D_{adversary_{y \to y_t}}|}{|D_{adversary}|} \approx 1, \forall (x,y) \in D_{adversary}, f_\theta^* = y_t \neq f(x) \tag{27}$$

3. *$D_{adversary}$ does **not** appear in the natural data:*

$$D_{adversary} \cap (D_{train} \cup D_{test}) = \emptyset \tag{28}$$

Table 3 presents an overview of the attack types covered by related work. It is noticeable that there is little previous work about defending against backdooring attacks, and no work at all that allows to use secure aggregation and thwart any kind of model poisoning. Badgasaryan et al. state that secure aggregation impedes all possible anomaly detections, as no information of a local client is accessible by the aggregator [7].

In this work, we focus on *semantic backdoors* (Definition 4), as it represents the most advanced attack considering the difficulty to detect such attacks compared to label-flipping attacks and the effort necessary by the attacker to craft a semantic backdoor compared to a trojaned backdoor. As just a very specific subset of data is misclassified, standard accuracy checking does not show noticeable abnormalities. Furthermore, the attacker can use natural data and does not need to modify samples to create training or test data which reduces his effort to craft such an attack.

Our goal is to defend semantic backdoors in federated learning while being able to provide a high level of privacy for the clients' data through a secure aggregation of local updates.

4.2. Adversarial Objectives & Threat Model in Federated Learning

Although backdoors can not only be injected in machine learning models in federated learning, its distributed characteristic, combined with the non-IID data distribution, makes it especially vulnerable to them. In the following section, we define the objectives of an adversary for injecting a backdoor into a machine learning model trained with federated learning.

In a poisoning attack that tampers with the training process in federated learning so that the classifier exhibits backdoor behavior, the primary goal of the adversary is to

		Federated Learning		Distr./ Outsour.	Others
		with SA	w/o SA		
Semantic Backdoor	Attack		Bagdasaryan et al. [7], Bhagoji et al. [25]		
	Defense				
Trojaned Backdoor	Attack		Bagdasaryan et al. [7]	*BadNets* [71], Tan et al. [69]	Liu et al. [68]
	Defense			*STRIP*, Liu et al. [83] [76]	Chen et al. [77], *Neural Cleanse* [80]
Label Flipping	Attack				
	Defense		*FoolsGold* [72], *SLSGD* [74]	*Auror* [75]	
Byzantine Tolerance	Attack				
	Defense			*Krum* [62]	(McMahan et al. [84])

Table 3.: Overview of related work

inject an effective backdoor, where "effectiveness" is typically measured in terms of the classifier's backdoor accuracy (cf. Equation (18)).

A secondary, but equally important objective is to ensure that the backdoored behavior of the model remains undetected. This observation motivated Bagdasaryan et al. [7] to list it as one of the adversarial goals to preserve good accuracy on the classifier's main task, and similarly motivated Bhagoji et al. [25] to evaluate a series of backdoor attacks in terms of achieved backdoor accuracy and stealth.

Given this, one may be tempted to define the goal of a backdoor attack as achieving simultaneously:

1. **Objective:** High backdoor accuracy $acc^{bd}_{X^*,y_t}$
2. **Objective:** High accuracy $acc_{D^{test}}$ on the main task (which we refer to as *main task accuracy*)

Note, however, that while preserving a high main task accuracy is necessary for the adversary to avoid detection — as accuracy testing provides a simple approach to detect model poisoning — nothing guarantees that it is also sufficient. It follows that another objective of an attacker should be, besides reaching high backdoor accuracy, to remain undetected in the realistic application scenario which deploys the model:

3. **Objective:** Imperceptibility through honest clients and the server

Due to the dynamic nature of the federated learning process, and in particular to the clients' contributions that repeatedly refresh (and hopefully improve) the global model, a highly accurate backdoor is likely to be erased by honest updates and quickly "forgotten" by the model. Therefore, the backdoor accuracy alone is not sufficient to quantify the effectiveness of the attack in the federated learning setting, and the durability of a backdoor must also be considered for a meaningful assessment of the adversarial success [7]. We can then define an additional adversarial goal as follows:

4. **Objective:** High durability of the backdoor

4.2.1. Threat Model.

Federated learning includes a central party, the aggregator S, and a set of clients. These clients can possibly be corrupted and controlled by an attacker to inject a semantic backdooring functionality. We assume that an attacker A can control up to $m \ll K$ clients. For each client, he can control all aspects of the local training process including, for example, the objective functions, training and test data, batch size, number of local epochs, etc. It follows that he is able to push an arbitrary update to the server. Additionally, an attacker-controlled client can also behave benignly in some iterations and maliciously in others to prevent detection. In contrast, he cannot influence the random selection of the clients and the averaging of the updates.

4.3. Requirements

In the following, we summarize the requirements in terms of security, privacy, and general functionality for a defense against backdoor attacks in federated learning.

4.3.1. Privacy

A defense system should achieve at least the same level of privacy as provided by plain federated learning. Additionally, integrating the secure aggregation of the clients' updates would bring a considerable advantage in terms of privacy protection. Thus, we define the following two privacy requirements:

- **P-1 Privacy of User Data:** All data should remain locally at the clients' devices.
- **P-2 Secure Aggregation:** To prevent the server from accessing individual updates, it should use the privacy-preserving secure aggregation protocol by Bonawitz et al. [4].

4.3.2. Security

We define the following three security goals of a defense by denying the adversarial objectives:

- **S-1 Weak Backdoor:** The poisoned model should achieve low backdoor accuracy.
- **S-2 Detectable Backdoor:** A backdoor should be detected if an aggregated update was poisoned.
- **S-3 Low Durability of the Backdoor:** The effect caused by a backdooring attack should vanish quickly.

4.3.3. Functionality

Despite the enhanced level of security, the federated learning with the integrated defense system should still achieve a comparable classifier performance with justifiable overhead in terms of communication and computation for the training process. Namely, we identify the following three functional requirements:

- **F-1 High Main Task Accuracy:** Let f_θ denote the normal classifier and $f_\theta^{defense}$ the classifier with integrated defense system, then the final main task accuracy should not be considerably lower than without the defense.

$$acc_{D^{test}}(f_\theta) \approx acc_{D^{test}}(f_\theta^{defense}) \tag{29}$$

- **F-2 Low Computation Overhead:** The classifier $f_\theta^{defense}$ that integrates a defense system should not add significant time overhead compared to the normal classifier f_θ.
- **F-3 Low Communication Overhead:** The classifier $f_\theta^{defense}$ that integrates a defense system should not significantly increase the amount of required messages compared to the normal classifier f_θ.

4.4. System Overview

To defend against model poisoning in federated learning with secure aggregation, we propose three techniques to detect malicious contributions. These techniques rely on statistical methods, neural-network activation clustering [77], and a feedback mechanism on the model update provided by clients respectively. The three defensive methods can be operated individually, and also be flexibly combined to make the system more robust.

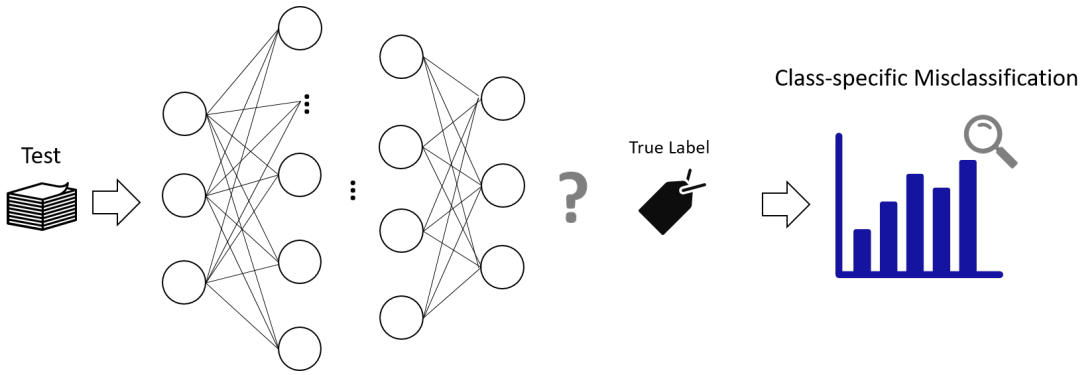


Figure 8.: Overview of defense layer #1: class-specific misclassification distribution analysis

4.4.1. Class-specific Misclassification Distribution

Our first defensive layer uses statistical methods to detect attempts to backdoor the global model. Our method consists in observing the behavior of the model on test data by inspecting the empirical distribution of the model's predictions on misclassified samples in a *class-specific* fashion. Figure 8 presents an overview of the technique. It relies on the expectation that the fraction of samples misclassified as belonging to a given class is roughly uniformly distributed among classes. Similarly, the fraction of samples belonging to a given class which is misclassified by the model is assumed to be roughly the same across subsequent training rounds regardless of the class. In contrast, a backdoored model should be slightly biased towards one class, and hence its class-specific misclassification rate exhibits a "non-uniform" distribution.

To be more specific, our detection method inspects the empirical distribution of class-specific misclassification rates and raises a warning if the results indicate suspicious behavior of the model. To capture the properties of "clean" and "suspicious" misclassification rates, we select suitable metrics that properly reflect the above-mentioned behaviors, allowing us to differentiate clean and poisoned models. We design these metrics in such a way that they summarize the misclassification rates for all classes in one number while amplifying the relevant information to detect suspicious behavior (note that standard error rates do summarize these rates, but at the price of losing information).

The first metric expresses the mutual differences among the error rates in a given round, and aggregates these differences to derive an "aggregated distance" — for various distance metrics, as we expand below. The second metric also uses differences among misclassification rates. Additionally, it compares these values with the results obtained from the most recent rounds.

Before introducing the details of the metrics, we define further notation to describe our

solution in a compact way. Given a classifier f_θ and two distinct labels $y_s, y_t \in Y, y_s \neq y_t$, we write $D_{y_s \to y_t}(f_\theta)$ to denote the set of instances belonging to a *source* class y_s that f_θ erroneously predicts as belonging to a *target* class y_t. For the sake of generality, we write $D_{y_s \to *}$ for the set of all misclassified y_s-instances in D, and similarly we denote by $D_{* \to y_t}$ the set of all instances in D which are misclassified as y_t-instances (we also omit an explicit dependency of the classifier to ease readability). Correspondingly, we denote the error rate of f_θ in misclassifying y_s-instances, as:

$$\mathrm{err}_D^{y_s \to *}(f_\theta) = \frac{|\{(x,y) \in D_{y_s} : f_\theta(x) \neq y_s\}|}{|D_{y_s}|}. \tag{30}$$

Similarly, we set the error rate of f_θ in misclassifying samples in D as y_t-instances, as:

$$\mathrm{err}_D^{* \to y_t}(f_\theta) = \frac{|\{(x,y) \in D \setminus D_{y_t} : f_\theta(x) = y_t\}|}{|D \setminus D_{y_t}|} \tag{31}$$

For the sake of readability, when the underlying data set D and the model f_θ are clear from the context, we denote the quantities of Equation 30 by $e^s_{f_\theta}(y_s)$ and $e^s(y_s)$, and by $err^t_{f_\theta}(y_t)$ and $err^t(y_t)$ the target-centric error for Equation 31.

In the following, we describe our metrics to identify backdoor injection attempts by analyzing the *class-specific* misclassification rates just defined. As our defenses are designed for a federated learning scenario, we denote by G^{t-1} the "previous" global model, i.e., prior to applying the round updates, and by G^t the "current" one (cf. Section 2.3).

Metric #1: Pairwise misclassifications among classes. Through extensive experiments, we observe that in each round t, while a clean model G^t presents class-specific misclassification rates that are relatively close to each other, a backdoored model tends to favor one class over the others, which leads to a misclassification distribution that exhibits peculiar behavior. We choose the first metric based on this observation. Namely, we analyze the model's misclassification behavior by inspecting the "distances" among class-specific error rates. Concretely, we calculate pairwise differences, over all pairs of distinct classes, among class-specific misclassification rates, and then aggregate the values. This metric summarizes all per-class rates and, at the same time, makes significant differences among these rates visible. In particular, we consider four aggregation rules for the pairwise differences between (source/target) misclassification rates for all distinct classes. We call them "(plain) sum", "absolute", "euclidean", and "squared

euclidean" norms denoted with the shortcuts: SUM, ABS, ED, SE.

$$\texttt{SUM} = \sum_{y \neq y' \in Y} \left(e(y) - e(y')\right), \tag{32}$$

$$\texttt{ABS} = \sum_{y \neq y' \in Y} \left|e(y) - e(y')\right|, \tag{33}$$

$$\texttt{ED} = \sqrt{\sum_{y \neq y' \in Y} \left(e(y) - e(y')\right)^2}, \text{ and} \tag{34}$$

$$\texttt{SE} = \sum_{y \neq y' \in Y} \left(e(y) - e(y')\right)^2. \tag{35}$$

Metric #2: Misclassifications among subsequent rounds. In our experiments we also observe that honest updates do not significantly affect the class-specific misclassification rates of the global model across subsequent rounds. In contrast, a freshly injected backdoor typically boosts the misclassification rate of one class. Consequently, we design our second metric based on these observations. We define slightly adapted versions of the aggregation rules above (SUM, ABS, ED, SE) to capture the differences of the class-specific error rates for the current (G^t) and the previous model (G^{t-1}), namely:

$$\texttt{SUM} = \sum_{y \in Y} \left(e_{G^t}(y) - e_{G^{t-1}}(y)\right), \tag{36}$$

$$\texttt{ABS} = \sum_{y \in Y} \left|e_{G^t}(y) - e_{G^{t-1}}(y)\right|, \tag{37}$$

$$\texttt{ED} = \sqrt{\sum_{y \in Y} \left(e_{G^t}(y) - e_{G^{t-1}}(y)\right)^2}, \text{ and} \tag{38}$$

$$\texttt{SE} = \sum_{y \in Y} \left(e_{G^t}(y) - e_{G^{t-1}}(y)\right)^2. \tag{39}$$

To determine the values for the two metrics, we assume that the server holds a test set D_{test} at this stage that it uses to analyze a new updated model.

The concrete steps are as follows: Firstly, the server takes the updated model and inputs its test set D_{test} (1st step). Afterwards, it determines all samples that are misclassified (2nd step) and sorts them according to *target*/*source* classes (3rd step). In step 4, it evaluates the metrics as described above for the current round. After determining these values for a round, they are compared to the values for the respective metric and aggregation rule of the previous r rounds. We set a "threshold" to distinguish between benign and malicious updates for every round. The analyzing party — the server — takes the mean and standard deviation of the metrics with respect to its test set D_{test} from the previous r rounds. r can be set according to the data set (i.e., overall size, amount of samples per class, etc.). We call this range of the last r rounds a "sliding

window", because it is moving in each iteration. If the value for the new model lies in a range of $[\mu - z\sigma, \mu + z\sigma]$ it is accepted, otherwise the server rejects the update (5th step). z is also selected based on the data set. Hence, the threshold is $\mu \pm z\sigma$.

4.4.2. Activation Clustering

The second defense applies activation clustering [77] to detect backdoored models. The underlying idea is that a backdoored update can be detected by inspecting the neural network activations of the last hidden (dense) layer when processing test samples. In general, activations explain the decisions made by a neural network. They extract high level features from the input samples that allow the network to distinguish between samples from different classes.

The key assumption is that benign samples and malicious samples that are predicted to the same class are expected to present notably different activations as their true classes are different. Therefore, supposing that the test set D_{test} contains backdoor samples (i.e., $X \cap X^* \neq \emptyset$ in the notation of Section 4.1), the activations of a backdoored model should exhibit a peculiar pattern on the target class that allows detection. However, if the test set D_{test} does not contain samples that have the specific characteristics of the backdoor, we will only have "normal" data predicted to the target class and therefore, we cannot detect the aforementioned differences in the activations between backdoor data and clean samples. We believe it is reasonable to assume that at least a few samples with the specific characteristics appear in D_{test}, because we are investigating semantic backdoor attacks, which means that these samples appear naturally in data and are not specifically crafted.

Concretely, in each round of federated learning, after receiving a new updated model the server classifies its test set D_{test} with it (1st step). It extracts the activations after the last hidden layer in the 2nd step, and groups it according to the predicted classes (3rd step). Afterwards, a dimension reduction technique such as PCA or ICA (cf. Section 2.2.3) is applied to the activations of each class (4th step) and the reduced activations are clustered into 2 groups with K-means (cf. Section 2.2.4) or an agglomerative clustering technique (5th step, cf. Section 2.2.4). Figure 9 summarizes these steps.

We design three metrics that assess how well these clusters fit to the data to decide if the model was poisoned. When a class is poisoned, which means it was successfully targeted by a backdooring attack, we expect that the clustering will divide the malicious and benign activations such that the two clusters will fit well, whereas when there is no poisoning the clustering splits the data at random somewhere. The reason is that, activations that belong to true samples of a class should be similar to each other, as they contain the same characteristics that link them to the same class. In contrast, backdoor images significantly differ from these "normal" samples and are only predicted

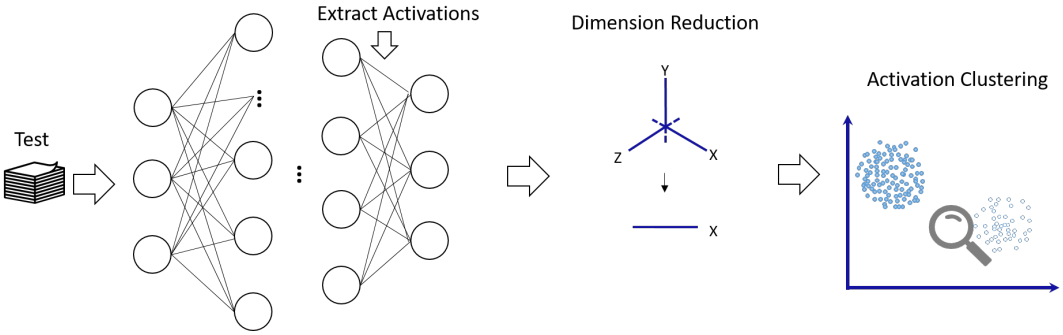
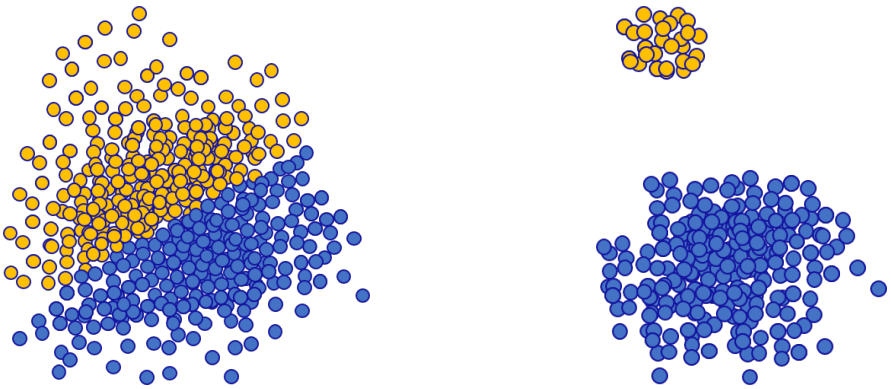


Figure 9.: Overview of defense layer #2: activation clustering analysis

to this class because of the backdoor. Figure 10 visualizes these expected behaviors. If this assumption is true, the phenomenon should, for example, be measurable in the clusters' sizes, the distances between the clusters' centroids, and the silhouette score as we will explain below. We choose these three measures for our experiments inspired by the approach of Chen et al. [77].



(a) Clean class **(b)** Poisoned class

Figure 10.: Expected results after activation clustering

Concerning the **clusters' sizes**, in the benign case we expect to receive two almost equally sized clusters for one class. The reason is that, since all samples have common characteristics that naturally let the model predict them into the same class, the corresponding activations should also exhibit these similarities such that the two clusters do not have a prominent boundary. In contrast, the expectation for the poisoned model is that the backdoored images have a common characteristic which maps them into the same (target) class, while they are, generally, inherently different from the natural images of that class. Therefore, these different characteristics should be reflected in the clustering through two clearly separated clusters. As the backdoor samples represent just a small subset of all samples predicted to belong to this class, the group of poisoned

samples will be significantly smaller given the previously described clustering behavior. If they are not only a small subset, we would be able to detect the poisoning directly via the main task accuracy and we would not need a more advanced defense.

The same reasoning is also valid for the **centroids' distances**. In the benign case, the two centroids (cf. Section 2.2.4) should be relatively close, because the data in the two clusters is similar. In the malicious case, we expect a larger distance between the two centroids because the main characteristics that cause the classification into the target class are different.

To use the **silhouette score** (cf. Section 2.2.4) to determine if we have a cluster with poisoned data, we calculate the class-wise average silhouette score. As stated before, we expect that poisoned classes will clearly split into two clusters: one cluster will contain the normal samples of the class, while the other cluster should contain the backdoor images. Consequently, the silhouette score of each cluster should be high. When the class is clean, there is no clear separation between the benign images that is visible through the clustering. In this case, our expected result is a small silhouette score.

For this defense we plan to determine a threshold based on empirical evaluation. Similar to Chen et al. [77], we expect to observe a boundary line for each of the three previously defined measures, where the benign cases will be below and the poisoned above (or vice versa). An active defense system can consider the boundary line as a threshold to distinguish poisoned updates from clean ones.

Compared to Chen et al., our design for the second defense aims to defend against semantic backdoors instead of trojaned backdoors. Additionally, we aim to include the analysis of the distances between the centroids, which is not proposed by Chen et al. Furthermore, we extend it to a real-time defense that can be applied during and not only after the training process. Moreover, we transfer the defense to federated learning, including applying it in a distributed manner that will be introduced in Section 4.4.3.

4.4.3. Feedback Loop

The third defense relies on honest clients supporting the server to detect poisoning attempts on federated learning. We call this layer *feedback loop*, because in every round the server asks clients to provide feedback on the current model. In principle the clients could use any detection method. Here we use the two previously introduced defense layers as Figure 11 shows.

We emphasize this approach has a very different spirit compared to previous proposals for defending federated learning against malicious behavior: beyond relying on clients for training the model, it leverages the diversity of clients' data for detecting malicious behavior.

More to the point, in every round of federated learning the server randomly selects a

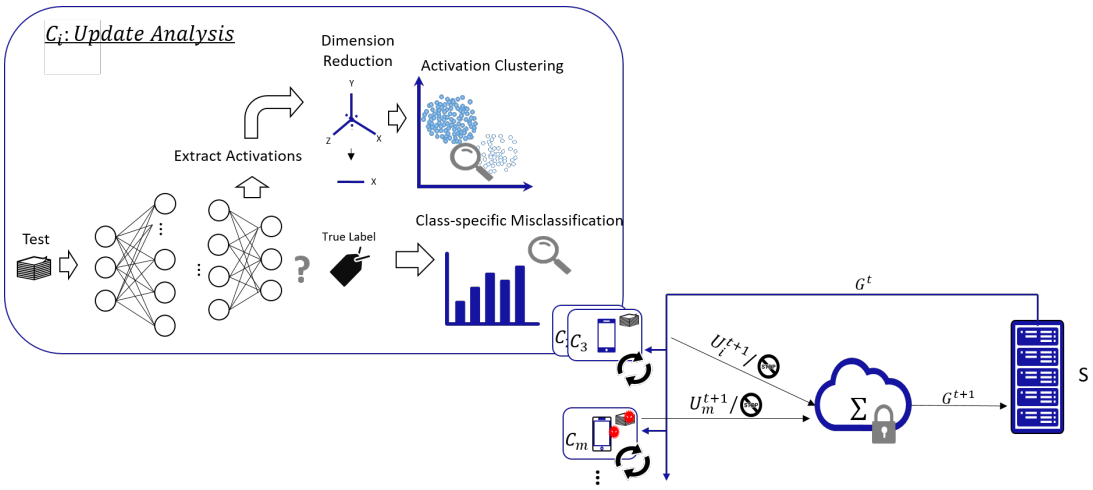


Figure 11.: Defending against semantic backdoor injection in federated learning with a client-driven feedback loop

subset of clients that do not only train local updates for a new global model, but also test the latest global model proposed in the previous round. Through combined testing and training it is possible to avoid additional communication overhead compared to asking two different sets of clients for updating and testing. If testing and training was split up, this would mean to, first, select a set of clients that will receive the new and previous global models for analyzing the recent update. Afterwards, another communication round would be required to transmit the accepted global model (or the previous global model after reverting) to the clients that were selected to provide the new local updates.

To concretely instantiate this defense layer, the aggregator needs to specify a detection technique that the clients apply to the current model. The naive option would be to test the accuracy of the model on the clients' local training data. Since main task accuracy testing is ineffective for detecting backdoor attacks, because the attacker can craft updates that preserve good main task accuracy [25], we apply the more fine-grained detection methods introduced in Section 4.4.1. We let the clients analyze the class-specific misclassification distribution and perform activation clustering. For the comparison with the previous r models, the selected clients will also be supplied with the previous r global models next to the recent global model by the server, because only a subset of clients participate in every round, and therefore they might have been chosen the last time a long time ago. Then, they would not be able to sufficiently evaluate a new update. When $e \leq K$ clients in an iteration have marked an update as probably poisoned, the server will accept a global model and aggregate the new local updates for the next global model.

Algorithm 4 presents a feedback loop with the two defensive techniques introduced

above used at a client. An aggregator S sends the new global model G^t together with the last r global models $G^{t-1}, ..., G^{t-r}$ to a client which responds either with a flag when its analysis suspects the new update to be poisoned, or with a new local update L_k^{t+1}. Based on all responses the server will either revert the global model G^t to G^{t-1}, if at least e clients think it was poisoned, or it will update the model to $G^t + \eta \sum_{i=0}^{C \cdot K} \frac{n_i}{n} L_i^{t+1}$.

Every selected client first analyzes the activation clustering, and if the result is not suspicious, it assesses the class-specific misclassification distribution. Only if both results are negative, the client will train a new local update and return it to the server. We will call it an *AND-combination* if we require both analyses to be passed. Alternatively, the order of the defense layers can be changed, other layers can be added, or an *OR-combination* can be performed meaning that both defense layers are executed in parallel and the update will be accepted if one of the two marks it as benign.

Note that this overview does not integrate the secure aggregation protocol. It can easily be added by letting only the clients that accepted the update engage in the secure aggregation protocol for a new update. All other clients will be considered as drop-outs, so they will be treated as clients that do not respond anymore (cf. [4]).

The feedback loop can be considered as an extra defense layer that a server deploys next to the defenses that it locally executes. The advantage of having the global model evaluated by clients is conceptually similar to the spirit of decentralized training that leverages broader and more diverse data that the clients can provide together.

Of course the feedback loop is accompanied by the risk of giving "voting power" on the global model to adversarially controlled clients, who may not honestly perform the poisoning analysis and provide wrong results regarding their belief about the model being poisoned. Therefore, similarly to the case of outsourced machine learning, the server's strategy to weigh the clients' feedback needs to be carefully designed based on the assumed number of attackers and the data.

Algorithm 4 Defense layer #3: feedback loop at client k

INPUT: $D_{test_k}, D_{train_k}, G^t, G^{t-1}, ..., G^{t-r}, n$ ▷ test data, train data, global model, old models, number of classes

OUTPUT: $flag_{clus}, flag_{miscl\_target}, flag_{miscl\_source} || L_k^{t+1}$ ▷ result clustering check, result misclassifcation check or local update

$a\_collection$, $miscl\_samples\_target$, $miscl\_samples\_source = [[] for\_ \in range(n)]$

foreach $(x, true\_label) \in D_{test_k}$ **do**

  $a =$ flattened activations of the last hidden layers of $G^t(x)$

  $a\_collection[G^t(x)].append(a)$

  if $G^t(x) \neq true\_label$ **then**

    $miscl\_samples\_target[G^t(x)] += 1$

    $miscl\_samples\_source[true\_label] += 1$

$miscl\_rates\_target = []$

$miscl\_rates\_source = []$

foreach $i \in range(n)$ **do**

  $a\_collection[i] = reduce\_dim(a\_collection[i])$ ▷ dimension reduction

  $cluster\_result = cluster(a\_collection[i])$ ▷ activation clustering

  $flag_{clus} = analysis\_clustering(cluster\_result, G^{t-1}, ..., G^{t-x})$

  if $flag_{clus} == ok$ **then**

    $miscl\_rates\_target.append(\frac{miscl\_samples\_target[i]}{sum(miscl\_samples\_target)})$ ▷ miscl. distribution

    $miscl\_rates\_source.append(\frac{miscl\_sampleas\_source[i]}{sum(miscl\_samples\_source)})$ ▷ miscl. distribution

  else

    return $flag_{clus}$ ▷ return activation clustering flag & class

$flag_{miscl\_target} = analysis\_miscl\_distr(miscl\_rates\_target, G^{t-1}, ..., G^{t-x})$

$flag_{miscl\_source} = analysis\_miscl\_distr(miscl\_rates\_source, G^{t-1}, ..., G^{t-x})$

if $flag_{miscl\_target} == ok$ && $flag_{miscl\_source}$ **then**

  $L_k^{t+1} = update\_model(G^t, D_{train_k})$

  return L_k^{t+1} ▷ return local update

else

  return $flag_{miscl\_target}$|| return $flag_{miscl\_source}$ ▷ return misclass. distribution flag & class

5. Experiments

In the following chapter, we evaluate the proposed three defense layers against semantic backdoor attacks. Firstly, we introduce the data set that we use for testing our ideas to defend against model poisoning in federated learning. Afterwards, we summarize the implementations for model poisoning and for the different experimental steps to analyze the success of our three defense layers.

5.1. Setup

5.1.1. Data Set: CIFAR-10

CIFAR-10 is a well-known vision data set published by Alex Krizhevsky [66] that contains 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The training set consists of 50,000 colored images with 32x32 pixels and 24-bit color per pixel (3 color channels) while the test set has 10,000 samples. Both sets are identically split between the classes such that for each class there exist 5,000 training samples and 1,000 test samples. State-of-the-art algorithms with various preprocessing techniques can achieve a top-1 accuracy of 90 % and up to 99 % [86, 87, 88]. Figure 12 shows some samples from the data set.

5.1.2. Model & Parameters

For our experiments we choose a simple neural network similar to the model used by McMahan et al. [6] with two convolutional layers with rectified linear unit activation (ReLU) function and "same"-padding followed by max pooling and three dense layers. The model contains 797,962 variables which correspond to 3,191,848 bytes. Details of the model are depicted in Table 4. The architecture achieves 82 % accuracy with independent and identically distributed data as shown in Figure 13. This is sufficient for our purpose as we aim to demonstrate model poisoning and investigate possible defenses which does not require an extremely high performing model. Instead, a lower main task accuracy allows to hide the effect of misclassifications caused by poisoning, because it is possible to remain close to the previous main task accuracy in spite of the targeted misclassifications caused by the poisoning through balancing it with other new correct classifications. Therefore, it makes poisoning easier because it is harder to

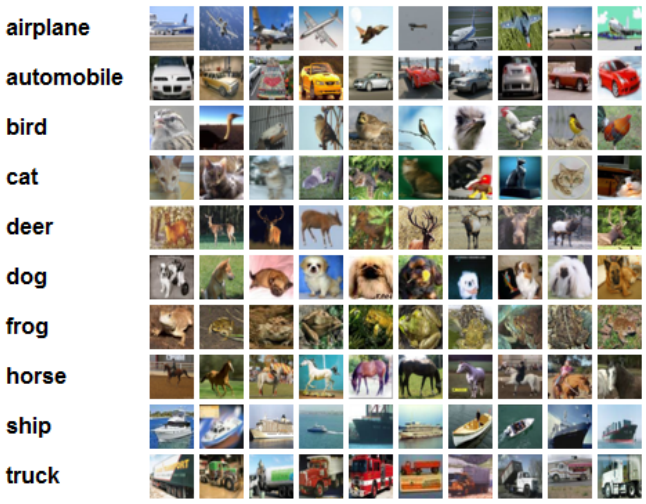


Figure 12.: Sample images from CIFAR-10 [5]

detect and defend. We use two local learning rates of 0.02 and 0.04 in our experiments with SGD (cf. Section 2.2.1) for optimization. Following prior work in [6], the global learning rate of the server is set to 1, and 10 of the 100 clients are chosen at random per round. Each selected client runs 5 local epochs with a batch size of 50 before reporting the update to the aggregator.

5.1.3. Data Preparation

For the learning process with federated learning, the distribution of the data among the clients has a significant impact on the training process. We call this distribution "data splitting". In the scope of this work, we investigate two possible data splittings in detail as it heavily affects the success of the learning [89].

In the beginning, we emulate independent and identically distributed data split between the clients by shuffling the training and test data, before distributing it evenly at random. Every client receives 500 training and 100 test samples.

As described in Section 2.3, data is typically non-IID in a federated learning setting. Thus, to emulate a more realistic setting, we distribute the data in an unbalanced manner (cf. Section 2.3) among the clients in a second set of experiments. To create unbalanced data, we divide both training and test data sets by applying the Dirichlet distribution [90] with $\alpha = 0.9$ as it is also done by Bagdasaryan et al. in [7].

For both data splittings we apply data augmentation to enable a better generalization of the model. First, we randomly crop the image to 24×24 pixels, before randomly flipping, adjusting the brightness, and the contrast at random as suggested in the

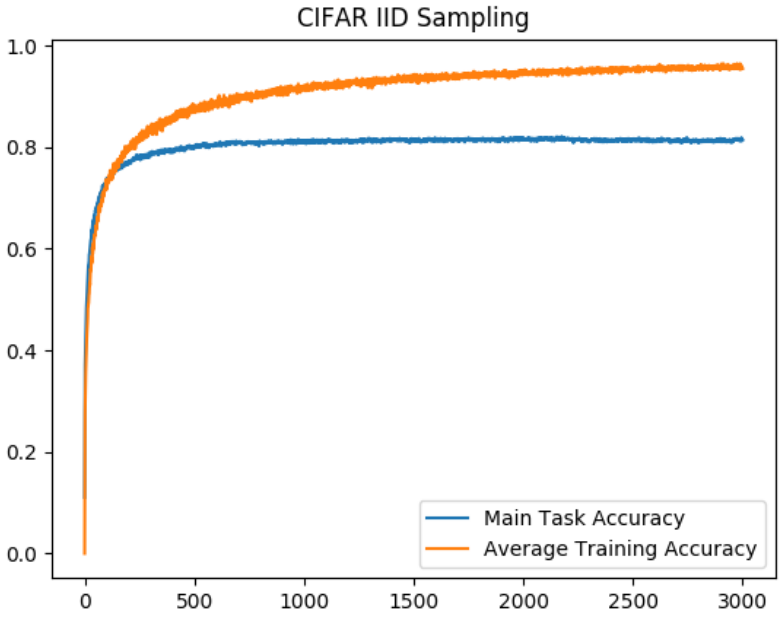


Figure 13.: CIFAR-10: Development of the main task accuracy of the convolutional neural network during a training process with federated learning

Name	**Type & Parameter Shape**	**# Trainable Parameters**
conv2d/kernel:	float32, 5x5x3x64	4800, bytes: 19200
conv2d/bias	float32, 64	64, bytes: 256
relu-activation	-	0
max_pooling2d (pool-size: 3x3, strides: 2)	-	0
conv2d/kernel:	float32, 5x5x64x64	102400, bytes: 409600
conv2d/bias	float32, 64	64, bytes: 256
relu-activation	-	0
max_pooling2d (pool-size: 3x3, strides: 2)	-	0
flatten	float32, 1600	0
dense/kernel	float32, 1600x384	614400, bytes: 2457600
dense/bias	float32, 384	384, bytes: 1536
relu-activation	-	0
dense/kernel	float32, 384x192	73728, bytes: 294912
dense/bias	float32, 192	192, bytes: 768
relu-activation	-	0
dense/kernel	float32, 192x10	1920, bytes: 7680
dense/bias	float32, 10	10, bytes: 40

Table 4.: CIFAR-10: Summary of the neural network model

Tensorflow Tutorial for CIFAR-10[1] and also used in [6]. Afterwards, all images are standardized.

5.2. Implementation

The experiments are implemented with Python on two servers. One is equipped with a Xeon E3-1240 V5 CPU with 8 vCore, 32 GiB RAM, 512 GB SSD, Ubuntu 16.04.6 LTS, and Python 3.5.2. The other has an Intel Core i5-9600k with a Geforce RTX 2070 WindForce GPU, 8 vCore, 8 GiB RAM, 120 GB SSD, Ubuntu 18.04.2 LTS, and Python 3.7.2. The latter one is used for all runtime measurements. Furthermore, we use the Tensorflow framework[2] with version 1.13.1 and CUDA version 10.0.130 for the GPU for implementing all machine learning algorithms. The underlying federated learning implementation is built on the LEAF-Benchmark for federated learning published by Caldas et al. in [91]. The LEAF-code can be found here: `https://github.com/TalwalkarLab/leaf`. We do not implement the secure aggregation protocol but as said before it can be easily integrated.

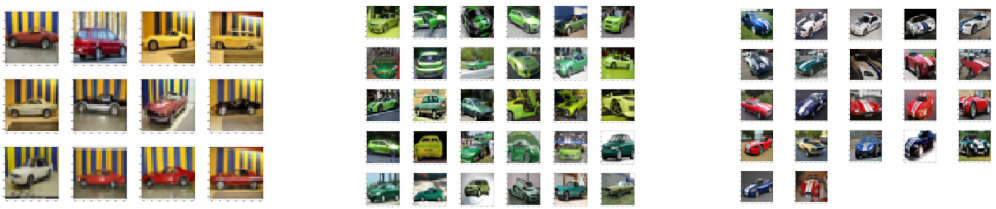
5.2.1. Poisoning Attacks

We use the same data as Bagdasaryan et al. in [7] to create semantic backdoors. They propose three different groups of images that share specific characteristics. In this case, we misclassify cars to birds. The first group of images to create a backdoor contains 12 samples of cars that are placed in front of a wall with vertical stripes as shown in Figure 14a. We will occasionally refer to them as the "wall"-backdoor later in this thesis. The second group contains 30 images of green cars (Figure 14b, the "green cars"-backdoor) and Figure 14c shows 22 samples of cars with racing stripes that form the third backdoor called "stripes"-backdoor. To inject a backdoor, one group of these images is used with a bird-label by a malicious client during its training. Generally spoken, when a backdoor is injected it aims to cause all samples with the specific characteristics to be misclassified to an attacker-defined class which is not the real class.

Basic Poisoning

Firstly, we follow Bagdasaryan et al.'s approach to inject a backdoor into the learning process. They pick a malicious client late in the learning process after 3,000 rounds when the model is converging. This client trains on 31.35 % malicious data which are images that contain the backdoor and 68.75 % normal data. Such a split ensures that the effect of the backdoor on the main task accuracy is reduced [7]. To get a sufficient

[1] `https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10`
[2] `https://www.tensorflow.org/`



(a) Striped wall **(b)** Green cars **(c)** Racing stripes

Figure 14.: CIFAR-10: Images used for injecting semantic backdoors

amount of training data, the number of images, that contain the required characteristic, is extended by randomly applying gaussian noise with $\sigma = 0.05$. Additionally to mixing benign and poisoned data, malicious clients double the local iterations and half the local learning rate to increase overfitting on the malicious data. After finishing the local training, the update is scaled such that it outvotes the other participants' contributions. Bagdasaryan et al. suggest $\frac{n}{\eta}$ as scaling factor, which corresponds to 100 for 100 clients (n) and the global learning rate $\eta = 1$. The test set for the backdoor accuracy is created out of three images chosen at random from the relevant backdoor image set while the others are used for training with malicious clients. The three images are extended to 1,000 by adding gaussian noise with $\sigma = 0.05$ to form a test set.

We observe that in contrast to Bagdasaryan et al.'s experiments the scaling of the attacker's update either causes a significant drop of main task accuracy or the backdoor is relatively weak and vanished just after one or a few more rounds. A significant drop in the main task accuracy would be easy to detect as it is normally not naturally observable in a benign learning process (compare to Figure 13). For example, Figures 15a and 15b show the results of short poisoning experiments with the green cars over 200 rounds with scaling factors 20 and 10. While scaling by a factor of 20 activates a higher backdoor accuracy, it also causes a drop of the main task accuracy by about 30 %. In comparison, the backdoor injected with the scaling factor 10 is less strong but also affects the main task accuracy just minimally. The effect of both backdoor injections vanishes quickly. The same effect can also be detected in Figure 15c that presents the injection of a "stripes"-backdoor in round 1,000 with a scaling factor of 10. It can be seen that it increases the backdoor accuracy for only one round significantly and after five rounds it drops back to a random level. This random level is another phenomenon that can be seen in the three figures. Some test samples are misclassified to the target class at random throughout the whole learning process. The same pattern can be found for the "green cars"-backdoor, but not for the "wall"-backdoor. This can be explained by the frequency with which these characteristics appear. Green cars are much more

common, while the stripes on the wall are very specific. Therefore, it is reasonable that with 80 % or less main task accuracy, more green cars will be misclassified as birds by chance than the cars with the stripes in the background.

Optimized Attack

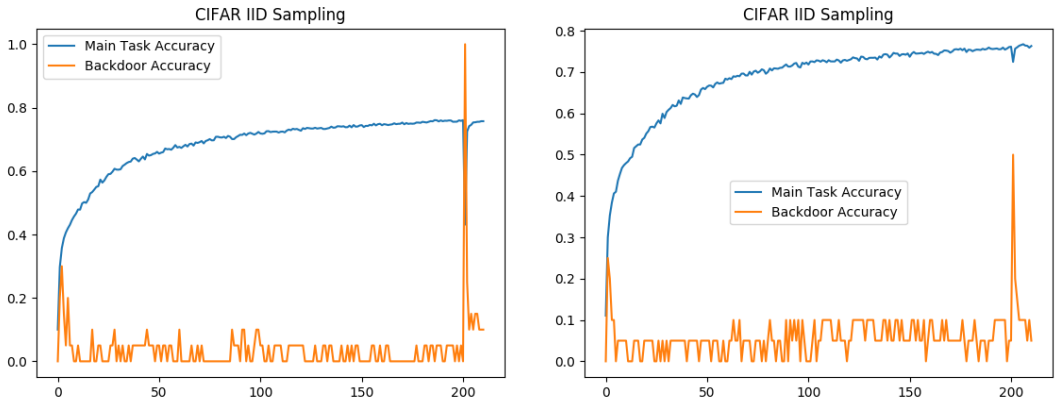
Based on these observations, we change the attack scenario slightly to achieve a more stealthy attack which does not considerably harm the main task accuracy. We pick a malicious client and inject a backdoor in every 10th round. In a group of 100 clients, when 10 clients are chosen every round, one client has a $\frac{1}{10}$ chance to be selected in a round. Hence, it is expected to be chosen on average in every 10th round.

Figures 16a to 16c show the resulting graphs with data split independently and identically distributed between the clients. The following three Figures 16d to 16f present the federated learning results with unbalanced client data. Our experiments indicate that unbalanced data causes much more fluctuation of the main task accuracy. Additionally, we also validate the natural expectation that it is easier to create a stealthy and more durable backdoor with unbalanced data, because unbalanced data means that clients provide less similar contributions such that the local updates vary more. It follows that even without malicious clients the main task accuracy in a benign training process fluctuates much more, when the data is distributed in an unbalanced manner compared to an IID splitting. Thus, it is easier for an attacker to influence the training into a targeted direction while remaining undetected ("stealthy"). The goal of an attacker is to achieve a high backdoor accuracy that remains as long as possible — it should be durable. In our experiments, we observe that backdoors, injected in a learning process with data distributed in an unbalanced manner, remain relatively long with a high backdoor accuracy.

To conclude, because non-IID data is typical for federated learning, it is more vulnerable to model poisoning than distributed machine learning between data centers where data is distributed in an IID manner. For the evaluation of our defense ideas we therefore focus on this optimized attack scheme with unbalanced data to have a realistic scenario for federated learning. The model achieves about 78 % main task accuracy in this setting.

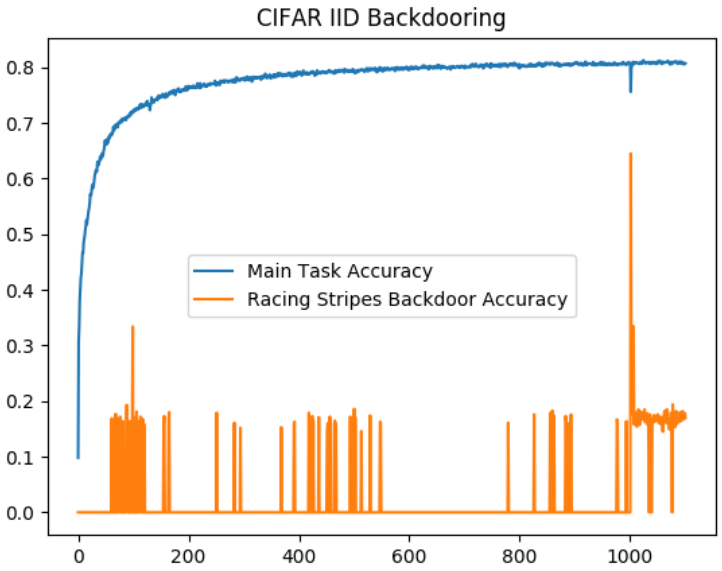
5.2.2. Class-specific Misclassification Distribution

The first defense idea we evaluate is the class-specific distribution of misclassification. After every round, different samples will likely be misclassified. We look into how these misclassifications change from two point of views as formalized in Section 4.4.1 with the two metrics. First, we analyze how the distribution of *source* classes — which are the



(a)

(b)



(c)

Figure 15.: CIFAR-10: Samples of single shot poisoning with semantic backdoors

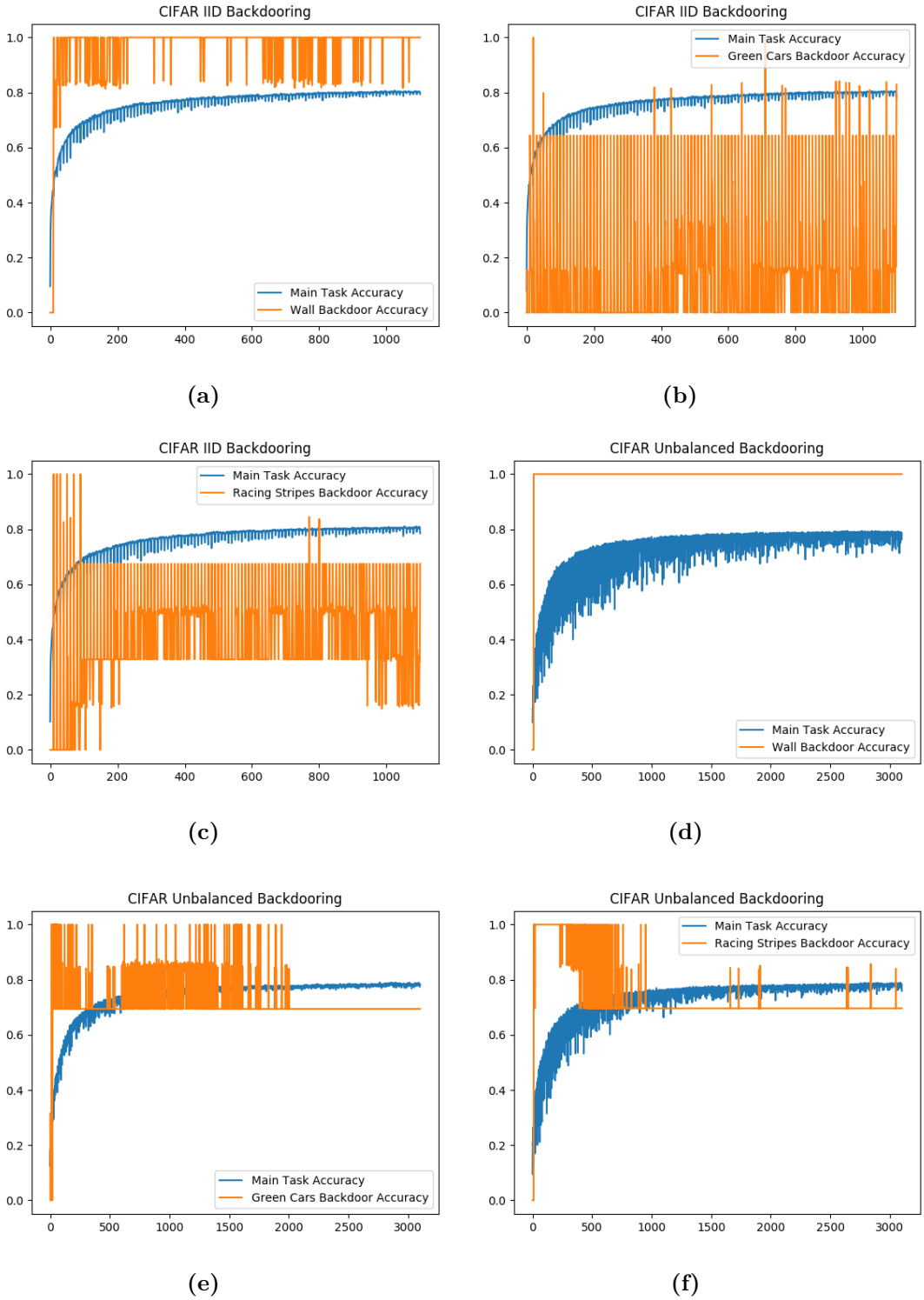


Figure 16.: CIFAR-10: Samples of repeated poisoning with semantic backdoors

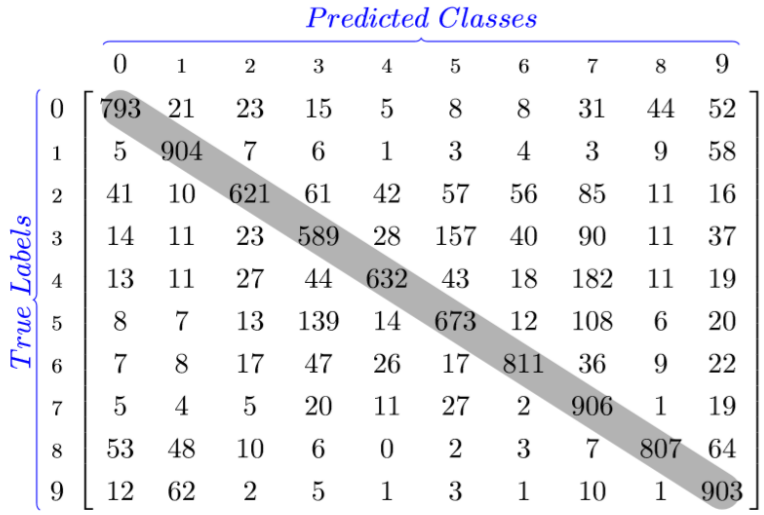


Figure 17.: Example of a confusion matrix. (For CIFAR-10 "0,1,2...,9" correspond to the classes "airplane, automobile, bird,..., truck".)

correct labels of the misclassified samples — changes ($D_{y_s \to *}$). Second, we consider the distribution of the *target* classes of the misclassified samples. *Target* classes denote the classes to which the misclassified samples are predicted ($D_{* \to y_t}$). We assume that the server has access to a test set, as for example the 10,000 test samples from CIFAR-10. In the following, we will present two types of experiments we conducted. The first type is called "static experiments". Their purpose is to get a first, rough understanding of how the statistical measures perform under various setup and parameters choices, and to further guide these choices into a "live" scenario. These "live" experiments are called active. They measure how the defenses perform in an active federated learning process under attack.

Static Experiments

In the first step, we run 12 poisoning experiments with data split in an unbalanced way between the clients over 3,100 rounds. In every 10th round a malicious client is chosen among 9 honest clients to provide a local update. The malicious client tries to inject its backdoor with a scaling factor of 10. After each round, we extract a confusion matrix of the test set and store it. Figure 17 shows an example of such a confusion matrix.

The first column contains the true labels and the first row represents the predicted classes. Hence, the highlighted diagonal contains the correctly classified samples. To assess the class-specific misclassification distribution of the *target* classes in a specific round, we sum up all elements of each column except the number of correctly classified samples (without the diagonal). For the *source* class misclassification distribution, we

do the same with the rows. Afterwards, we evaluate Metric #1 and Metric #2 for both *source* and *target* classes and with all four aggregation rules (cf. Section 4.4.1) and plot their development over the 3,000 rounds.

In the next step, we determine a threshold to distinguish benign from poisoned updates with a sliding window for the last z rounds. z is set to several values between 10 and 1,000. For the first two rounds, we calculate the values of the metrics and store them. Starting from the 3rd round for Metric #1 and 4th round for Metric #2, we calculate the mean μ and the standard deviation σ of the two metrics (with the different aggregation rules) of the last r rounds in a sliding window. A round will be marked as poisoned and not considered for the calculation of means and standard deviations in the later rounds if the metric of the round does not fall into the range $[\mu - z\sigma, \mu + z\sigma]$. We test values between 2 and 4 for z and determine the true positive and false positive rates based on the 12 experiments.

Active Experiments

Based on the results from the static evaluation, we pick the two best performing aggregation rules for each metric. In the experiments, we use a sliding window that considers the metrics' values for the server's test set in the last 10 rounds and we revert an update of the global model to the previous model when it is marked as poisoned. Each combination of a metric and an aggregation rule is run several times to determine representative true positive and false positive rates, as well as to evaluate the success of the defense with respect to the effectiveness of the backdoor and its effect on the main task accuracy.

5.2.3. Activation Clustering

The second defense idea that we test and analyze is the activation clustering. As described in Section 4.4.2, it extracts the activations of test samples of the last hidden dense layer after each round. After applying a dimension reduction technique, all reduced activations, that were predicted into the same class, are clustered. The expectation is that, given the special characteristic of the backdoor, for instance the color of the car, appears in the test data, it will cause a misclassification. The activations of these samples should significantly differ from the normal data of that class. If this expectation is true, the clustering should split benign and backdoored samples of a poisoned class (cf. Figure 10b). We again assume that the server holds a test set from which it can derive the activations after a learning iteration.

Static Experiments

To test whether it is possible to distinguish benign and malicious samples via activation clustering after injecting the "wall"-backdoor, we create 5 clean models, 3 of them over 1,000 rounds and 2 over 3,000 rounds. After the 1,000/3,000 rounds, we store the clean models ($acc^{bd}_{X_{adversary},2} = 0$) and inject the backdoor with the images that contain the striped background wall. The poisoned models are also stored afterwards such that we receive 10 models in total. They achieve a backdoor accuracy of about 59 %, 90 %, and 3x 100 %.

In the next step, we load the models and collect the activations of the last hidden layer of all samples of the test set and the backdoor images. Afterwards, a dimension reduction technique is applied on all collected activations to extract the most relevant components out of 192 values in the last layer. We reduce every activation to 3 to 15 dimensions in several experiments. For three dimensions, we plot the results. Following prior art [77], we apply Principal Component Analysis (PCA, cf. Section 2.2.3) and Fast-Independent Component Analysis (FastICA, cf. Section 2.2.3) as dimension reduction techniques. Our idea is that the activations of backdoored images are differently spread than the benign activations such that PCA is able to detect the important components where the two groups of images differ due to their variance. For ICA, we expect it to be useful to identify the parts of the activations caused by the specific characteristics of the backdoor as different signals than the normal parts. PCA finds uncorrelated principal components while ICA searches for independent variables, therefore the two techniques have different results. As both have interesting properties for our application, we test both to investigate which one is more suitable for our purposes.

After dimension reduction, the reduced activations are clustered. We use K-means (cf. Section 2.2.4) and four types of agglomerative clustering (ward, complete linkage, average linkage, and single linkage, cf. Section 2.2.4). The clustering algorithms are picked as they allow to specify in advance how many clusters k they will create. Because we aim to obtain two clusters, one with benign and one with poisoned data, this is a desirable property. As distance metric we use the euclidean distance.

For the implementation of the dimension reduction techniques, the clustering algorithms, and the silhouette score (cf. Section 2.2.4) we use the Python machine learning library scikit-learn[3].

After creating the clusters, we test the three measures clusters' sizes, the distances between the clusters' centroids, and the silhouette score to access how well these clusters fit to the data. As described before in Section 4.4.2, when a class is poisoned, we expect that the clustering divides the malicious and benign activations such that creating two clusters fits for the given data and they separate the data well. Whereas when there is

[3] `https://scikit-learn.org/stable/`

no poisoning the clustering should split the data at random somewhere and it would intuitively be more reasonable to create one instead of two clusters. For these three metrics we validate if we obtain the expected results when evaluating the 5 poisoned and 5 clean models such that they would allow us to distinguish benign and poisoned models by clustering and analyzing the clusters.

5.2.4. Feedback Loop

The final defense idea is to include the clients via a feedback loop into the update analysis as this would enable to renounce the local test of the server such that it not necessarily needs to have a test set anymore. To avoid additional communication overhead, the same clients that are selected for the new update iteration are also chosen for this feedback loop.

Concretely, they first analyze the global model they receive for the new training round. To ensure that all selected clients have access to the previous, most recent 10 models to compute the relevant statistics, the server sends not only the recent global model G^t but also the previous 10 ones. Each selected client calculates then the mean and the standard deviation of the metrics of the class-specific misclassification distribution based on these 10 models and checks if the value associated with the new update falls into the allowed range. If its analysis' result is positive, so that the model does not appear to be poisoned, it continues with the training phase to create a new local update. In the case of a negative outcome of the local analysis, which means that the client suspects the model to be poisoned, the client just returns its rejection of the global model and does not train. The server reverts the global model when at least $e \leq K$ of the clients picked for the update reject the model. e needs to be experimentally optimized. We fixed it to 3 for our experiments such that at least a quarter of the clients has to suspect a model to be poisoned to reject it. This enables already a few honest clients to mark an update as poisoned.

Additionally, we also evaluate the effect of combining the feedback loop with the class-specific misclassification distribution analysis with a local test set at the server as explained in Section 5.2.2. Thereby, we split the test data between clients and server 0/100, 25/75, 50/50, 75/25, and 100/0 in 5 configurations with 10 experiments each with the respective metric and aggregation rule that performs best for the server/the feedback loop. Please note that 100/0 is equal to the experiments described in Section 5.2.2 where the full test data is used at the server. 0/100 is the same as using all test data for the feedback loop without any analysis at the server as described in the previous paragraph.

After receiving the updates from the clients, a server first tests with its local test set the class-specific misclassification distribution, before starting a new training iteration.

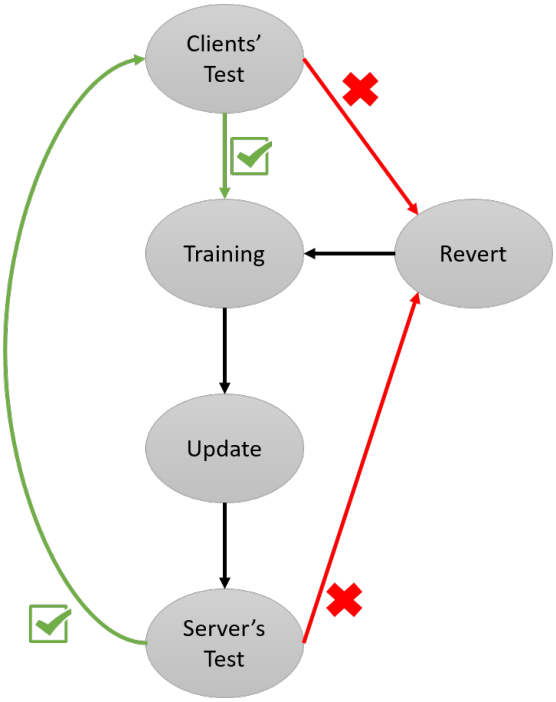


Figure 18.: Process flow of the combination of the local class-specific misclassification distribution analysis and the client-driven feedback loop

In a new training iteration, all clients use their test data to analyze the class-specific misclassification distribution. If enough clients accept the model and respond with new updates, the server aggregates them, and tests again with its data. Figure 18 pictures this process flow.

6. Results

In the following chapter, we present our empirical results of the tests of our three defense proposals against semantic backdoors in federated learning. We always include the true positive rate for detecting poisoned updates, the false positive rate, and the main task accuracy achieved when including the defense. A true positive rate means hereby the accuracy with which a "poisoned" round is correctly identified as poisoned and the false positive rate denotes the percentage of rounds that are marked as poisoned although they are benign. We refer to rounds in which the global model has been poisoned as "poisoned rounds". All values are averaged over several experiments, and we provide the exact number of experiments from which the average was derived. Additionally, we provide the standard deviation in parentheses after each averaged value. We also present measurements about the runtime overhead caused by our defense layers. We conclude the chapter with introducing a small extension for the defensive system and discussing the presented results based on the requirements defined in Chapter 4.

6.1. Class-specific Misclassification Distribution

6.1.1. Static Experiments

As described in Chapter 5, we first analyze the success of the defensive layers in a static setting. Static means here that we run a complete training process including the poisoning in every 10th iteration. Afterwards, we access the intermediate misclassification distribution of every training round and analyze it with the proposed metrics. The purpose is to get a preliminary understanding if the defense layer can work and if it is compatible to a threshold scheme for the active experiments.

In the case of the class-specific misclassification distribution analysis, we extract the confusion matrix of the test set in every round. Then, we test the effect of several sliding window sizes. As in the first rounds there is a lot of variation, we also evaluate the effect of starting the analysis later. For the window size we test values between 10 and 1,000, and fix it for the later experiments to 10 because our results indicate this to be the optimal window size. A bigger range decreases the false positive rates but also worsens the true positive rate. Table 5 shows an example of these effects.

When starting the analysis in a later round, as for example in round 1,000, we can

Distance Metric	Size: 10		Size: 100		Size: 1000	
	TP	**FP**	**TP**	**FP**	**TP**	**FP**
ABS	0.99132176	0.03894080	0.87116154	0.00519210	0.05006675	0.0
SUM	0.61081441	0.03931167	0.41388518	0.00222518	0.07343124	0.0
SE	0.98798397	0.09123275	0.88785046	0.03671562	0.06008010	0.0
ED	0.95794392	0.06082183	0.86114819	0.01149681	0.06008010	0.0

Table 5.: CIFAR-10: Static experiments - Example of the effect of different sliding window sizes on the Metric #1 of the class-specific misclassification distribution with respect to *target* classes and a tolerance range of 3σ

Distance Metric	2σ		3σ		4σ	
	TP	**FP**	**TP**	**FP**	**TP**	**FP**
ABS	0.9338889 (0.0564675)	0.9600309 (0.0951333)	0.6463889 (0.2541998)	0.0474691 (0.0104394)	0.4675000 (0.2815306)	0.0069753 (0.0029039)
SUM	0.9794444 (0.0208537)	0.9778704 (0.0156070)	0.3491667 (0.2934316)	0.0419753 (0.0091241)	0.2066667 (0.2458357)	0.0066667 (0.0031789)
SE	0.9477778 (0.0417296)	0.8680556 (0.0957862)	0.6769444 (0.2529509)	0.0830556 (0.0112797)	0.5597222 (0.2867360)	0.0263889 (0.0071324)
ED	0.9422222 (0.0399614)	0.9037346 (0.1176054)	0.6141667 (0.2666081)	0.0535494 (0.0061891)	0.4458333 (0.2994319)	0.0104321 (0.0031348)

Table 6.: CIFAR-10: Static experiments - Results of the Metric #1 analysis for the class-specific misclassification distribution with respect to *target* classes and a 10-rounds sliding window

achieve higher true positive rates than compared to starting the analysis for poisoned updates earlier or directly in the beginning. This is due to the high noise of the updates when a fresh model starts learning. Nevertheless, we aim to defend model poisoning from the beginning in the training process, because it is not realistic to start the defense late as the previous rounds would need to be guaranteed to be clean which is hard to achieve and to justify in practice unless training is done in a trustworthy environment. Therefore, we decided to start the later tests in round 3. It is the earliest possible moment, as we need at least two values to determine the standard deviation for setting our threshold range. It follows that we require these first iterations to be done in a trusted environment. Table 6 shows the averaged true positive and false positive rates for the class-specific misclassification distribution concerning *target* classes across 12 experiments. Tables 7 to 9 contain the respective results for Metric #1 for *source* classes and Metric #2 for both *target* and *source* classes, for the same 12 experiments. One complete model training with 3,000 rounds took on average 16.08 hours.

True positive rates should be as high as possible while the corresponding false positive rates should be minimal because discarding benign updates needs to be avoided. To understand which aggregation rules perform best, we calculate the complement of a mean false positive rate $(1 - FP)$ and sum it up with the corresponding mean true

Distance Metric	2σ		3σ		4σ	
	TP	**FP**	**TP**	**FP**	**TP**	**FP**
ABS	0.6572222 (0.2638702)	0.6529630 (0.3548107)	0.3630556 (0.3554899)	0.0025000 (0.0013950)	0.1019444 (0.1196016)	0.0001235 (0.0002310)
SUM	0.5952778 (0.2279232)	0.2186728 (0.2191390)	0.2411111 (0.2932428)	0.0021914 (0.0008876)	0.0980556 (0.2351968)	0.0000617 (0.0001380)
SE	0.5330556 (0.2857785)	0.1958025 (0.1598081)	0.3694444 (0.3452879)	0.0063580 (0.0022520)	0.1747222 (0.2363592)	0.0003395 (0.0005118)
ED	0.6663889 (0.2710250)	0.5375000 (0.3619705)	0.3100000 (0.3333222)	0.0029938 (0.0015003)	0.0558333 (0.0716747)	0.0001852 (0.0004141)

Table 7.: CIFAR-10: Static experiments - Results of the Metric #1 analysis for the class-specific misclassification distribution with respect to *source* classes and a 10-rounds sliding window

Distance Metric	2σ		3σ		4σ	
	TP	**FP**	**TP**	**FP**	**TP**	**FP**
ABS	0.8472222 (0.1563462)	0.2438889 (0.0169843)	0.6469444 (0.2645172)	0.0388272 (0.0074380)	0.3819444 (0.2346805)	0.0066667 (0.0019830)
SUM	0.9702778 (0.0175044)	0.9573765 (0.0201670)	0.6161111 (0.2814145)	0.0654012 (0.0537057)	0.3702778 (0.2720344)	0.0052160 (0.0034195)
SED	0.9480556 (0.0423272)	0.9101852 (0.0612997)	0.6333333 (0.1127436)	0.4118210 (0.0392588)	0.3980556 (0.1049820)	0.1581790 (0.0368832)
ED	0.9791667 (0.0415471)	0.9776543 (0.0411893)	0.9333333 (0.0551597)	0.9309877 (0.0558522)	0.4750000 (0.3444870)	0.4737963 (0.3445229)

Table 8.: CIFAR-10: Static experiments - Results of the Metric #2 analysis for the class-specific misclassification distribution with respect to *target* classes and a 10-rounds sliding window

Distance Metric	2σ		3σ		4σ	
	TP	**FP**	**TP**	**FP**	**TP**	**FP**
ABS	0.7786111 (0.2492487)	0.1372531 (0.0111033)	0.4744444 (0.2976337)	0.0089506 (0.0093975)	0.0922222 (0.1310805)	0.0008642 (0.0011449)
SUM	0.7833333 (0.2259302)	0.1453395 (0.0195084)	0.5747222 (0.3167089)	0.0136420 (0.0056430)	0.1902778 (0.3252533)	0.0010185 (0.0010380)
SE	0.7119444 (0.1533059)	0.4613272 (0.0295347)	0.4211111 (0.2533346)	0.1118827 (0.0214823)	0.2833333 (0.2536511)	0.0406790 (0.0497281)
ED	0.3966667 (0.2545803)	0.0794444 (0.0132028)	0.2719444 (0.2771130)	0.0163580 (0.0039016)	0.1197222 (0.1584442)	0.0060185 (0.0127228)

Table 9.: CIFAR-10: Static experiments - Results of the Metric #2 analysis for the class-specific misclassification distribution with respect to *source* classes and a 10-rounds sliding window

		2σ	3σ	4σ
ABS	Target	0.4869290	0.7994599	0.7302623
	Source	0.5021296	0.6802778	0.5509105
SUM	Target	0.5007870	0.6535957	0.6000000
	Source	0.6883025	0.6194599	0.5489969
SE	Target	0.5398611	0.7969444	0.7666667
	Source	0.6686265	0.6815432	0.5871914
ED	Target	0.5192438	0.7803086	0.7177006
	Source	0.5644444	0.6535031	0.5278241

Table 10.: CIFAR-10: Evaluation of Metric #1 based on the static experiments

		2σ	3σ	4σ
ABS	Target	0.8016667	0.8040586	0.6876389
	Source	0.8206790	0.7327469	0.5456790
SUM	Target	0.5064506	0.7753549	0.6825309
	Source	0.8189969	0.7805401	0.5946296
SE	Target	0.5189352	0.6107562	0.6199383
	Source	0.6253086	0.6546142	0.6213272
ED	Target	0.5007562	0.5011728	0.5006019
	Source	0.6586111	0.6277932	0.5568519

Table 11.: CIFAR-10: Evaluation of Metric #2 based on the static experiments

positive rate. The reason for complementing the false positive rates is that they should be as small as possible while the true positive rates should be as high as possible. If we complement the false positive rate which is in $[0, 1]$ with $1 - FP$, the resulting value should now also be large to indicate a good false positive value. It follows that the higher the combined value $TP + (1 - FP)$ the better the metric and aggregation rule performs in our static experiments. Tables 10 and 11 present the results. For Metric #1, ABS and SE for the *target* classes perform best when tolerating a range of 3σ around the mean of the last 10 rounds. For Metric #2, ABS and SUM for the *source* classes and a range of 2σ perform best.

Figures 19 and 20 show examples for the development of the metrics where the backdoor with the striped background walls is injected every 10th round. In regular distances a peak is visible. Zooming in shows that it happens exactly every 10th round, when the malicious client tries to inject its backdoor. This observation appears in all our experiments and it is the basis for the idea of our first defense layer. With mean and standard derivation we aim to detect these outliers. For instance, Figure 21 visualizes the Metric #1 analysis of one experiment with aggregation rule ABS and 3σ-range for the *target* classes. It can be seen that most injections (98.5 % in this experiment) are correctly detected.

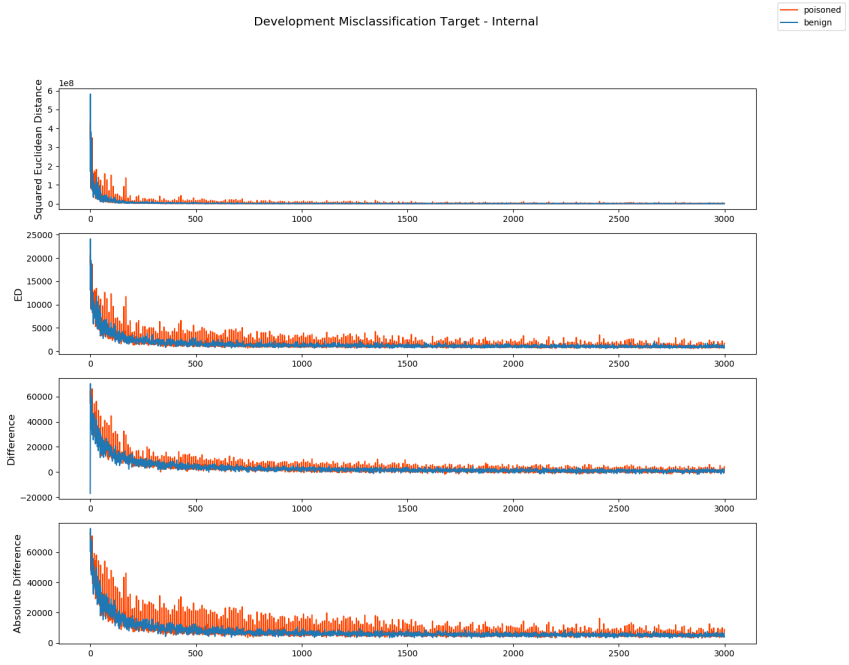


Figure 19.: CIFAR-10: Example for the development of Metric #1 for *target* classes while injecting the striped wall-backdoor

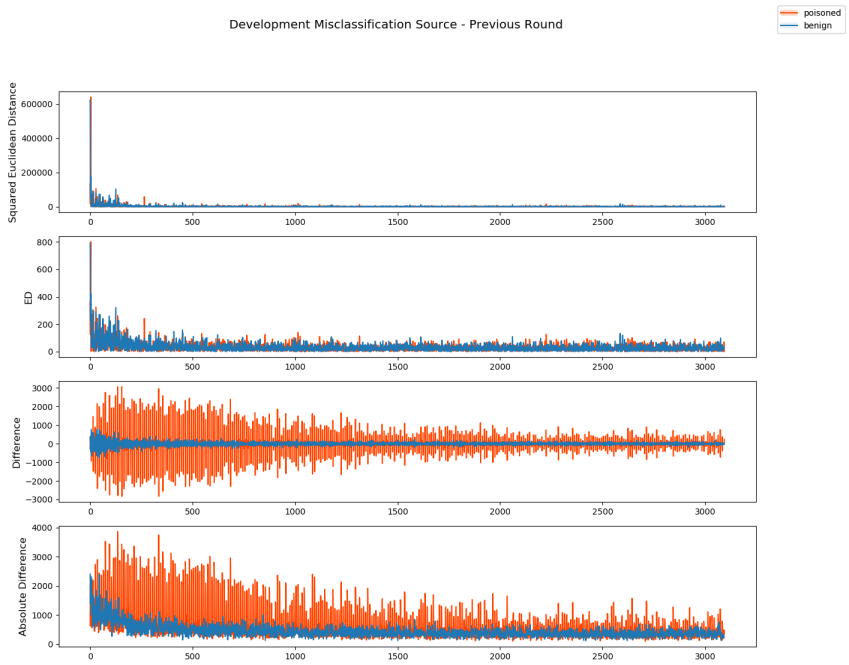


Figure 20.: CIFAR-10: Example for the development of Metric #2 for *source* classes while injecting the wall-backdoor

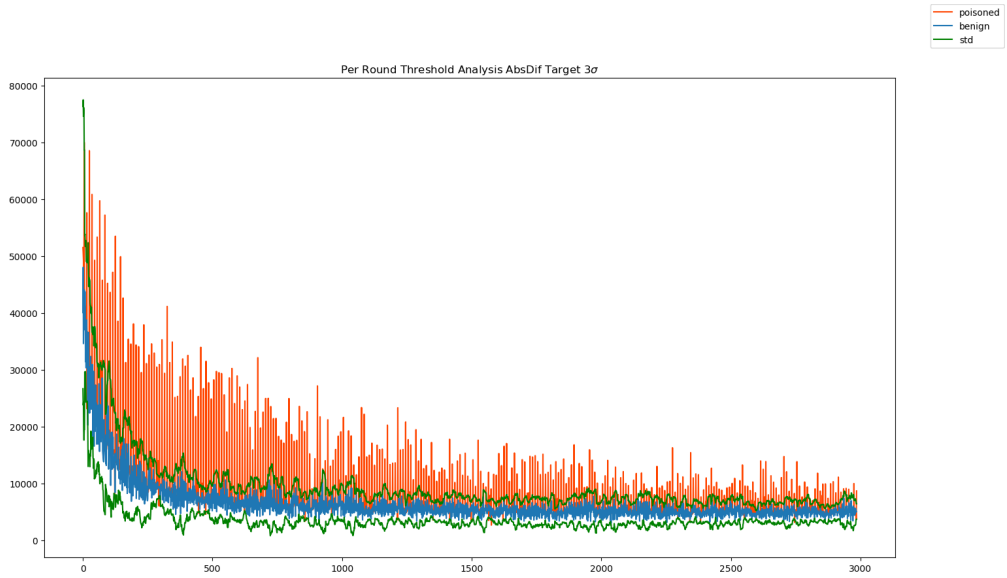


Figure 21.: CIFAR-10: Example for the effect of a Metric #1 analysis with ABS for *target* classes while injecting the wall-backdoor

6.1.2. Active Experiments

We use the insights about the best performing metrics in the static experiments for fixing the setup and parameters in the active learning experiments. A defense is considered as successful if the measured backdoor accuracy during the training process is normally 0 % and just occasionally a bit higher but still less than the random level (about 15 - 20 %). Experiments that could not successfully defend against the backdooring, i.e., the backdoor accuracy remains high across the learning process, are called "failures". The effect of a successful defense in an active learning process can be seen in Figure 22, while Figure 23 presents a failure.

Tables 12 to 14 present the results for Metric #1 and Metric #2. For Metric #1 with ABS, SE, and 3σ, 13 out of 18 experiments successfully defend the poisoning with high true positive rates above 90 % and with 8 % false positives while the 18 experiments retain an average high main accuracy around 75 %. The averaged values provided in Tables 12 to 14 are across all 6 experiments (1 failure) for SE and 12 experiments (4 failures) for ABS.

For Metric #2, we test ABS and SUM with a threshold range of 2σ because these aggregation rules perform best for Metric #2 in our static experiments. As the false positive rates are very high which reduces the average main task accuracy to about 69 % and 47 %, we decided to switch (after 7 experiments with ABS and 3 experiments with SUM) to the same aggregation rules but with the *target* classes and a range of

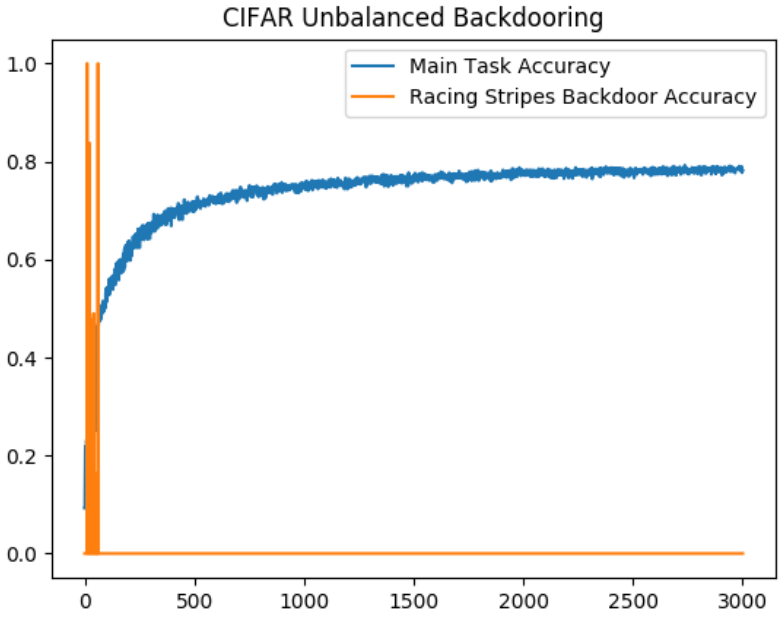


Figure 22.: CIFAR-10: Example for a successful defense with Metric #1 with *ABS* and 3σ for *target* classes while injecting the stripes-backdoor

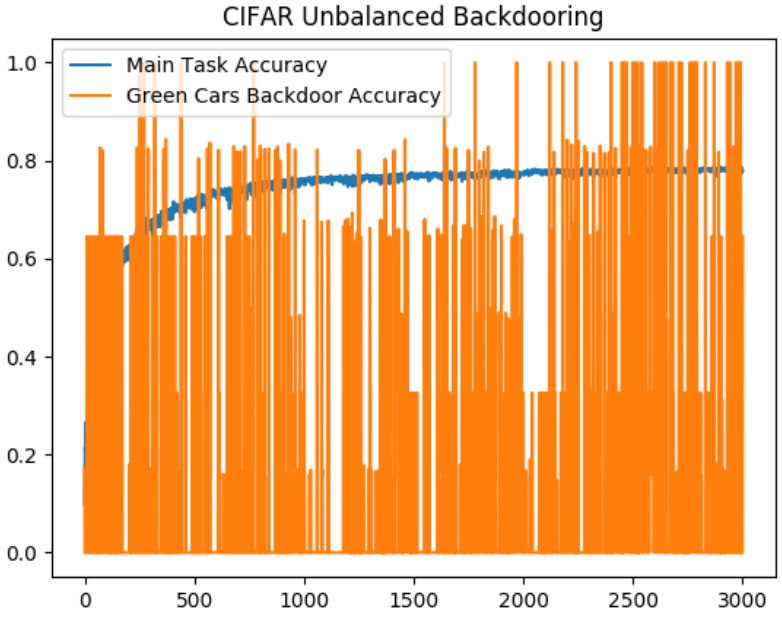


Figure 23.: CIFAR-10: Example for a failed defense with Metric #1 with *ABS* and 3σ for *target* classes while injecting the green cars-backdoor

Distance Metric	Metric #1 3σ, Target Classes			
	TP	**FP**	**Main Task Accuracy**	**Time (hours)**
ABS	0.86583325 (0.1758278)	0.1533641 (0.2696056)	0.78	16.15
SE	0.7694445 (0.2959385)	0.0941973 (0.0134433)	0.7791	16.2

Table 12.: CIFAR-10: Active experiments for class-specific misclassification distribution analysis - Results for Metric #1 with a 10-rounds sliding window

Distance Metric	Metric #2 2σ, Source Classes			
	TP	**FP**	**Main Task Accuracy**	**Time (hours)**
ABS	0.98 (0.0327570)	0.8172686 (0.0804515)	0.6858	16.25
SUM	0.9833333 (0.0288675)	0.9141977 (0.1010808)	0.4717	16.25

Table 13.: CIFAR-10: Active experiments for class-specific misclassification distribution analysis - Results for Metric #2 with a 10-rounds sliding window - Part 1

3σ, because they also show promising results in the static experiments. For *ABS*, we execute 12 experiments that reach an average main task accuracy of 78 %. 5 experiments are fully successful, 4 experiments reduce the backdoor accuracy at least by 50 %, and 3 experiments fail to defend against the backdoor injection. For *SUM*, 4 out of 6 experiments completely defend the backdoor injection, 1 is partially successful, and one experiment fails. An average main task accuracy of 79 % is achieved.

To conclude, Metric #1 with the parameters *ABS* and 3σ is most successful in defending against backdooring attacks in the active experiments. Except from the experiments with a threshold range of 2σ, the main task accuracy is not or just marginally affected.

6.2. Activation Clustering

6.2.1. Static Experiments

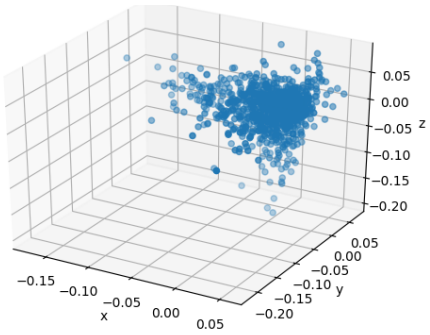
We first visualize the activations of samples that are predicted to belong to the poisoned class 2, in which the backdoor is injected, reduced to 3 dimensions to get a first impression if this second defense layer can work. Figure 24 shows the results with

Distance Metric	Metric #2 3σ, Target Classes			
	TP	**FP**	**Main Task Accuracy**	**Time (hours)**
ABS	0.6511112 (0.3203513)	0.0599383 (0.0159206)	0.7775	16.28
SUM	0.7794443 (0.3237929)	0.0579012 (0.0053400)	0.7857	16.28

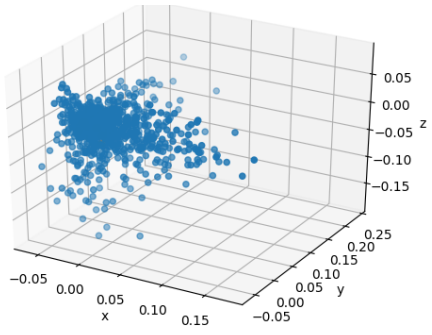
Table 14.: CIFAR-10: Active experiments for class-specific misclassification distribution analysis - Results for Metric #2 with a 10-rounds sliding window - Part 2

FastICA for the 6 models. The results for the other 4 models can be found in the Appendix in Figure 26. The left column presents the activations before poisoning the model, the right column shows the same model after the next training iteration where a malicious client was chosen for the updating process and injected its backdoor. The plots of the data after reducing the extracted activations from the same 10 models to 3 dimensions with PCA can be found in Appendix A. The reduced activations of the images that contain the striped wall are marked in orange. In the clean models, none of the images with the striped background wall are misclassified as birds, which is why they do not appear in the left images. For both dimension reduction techniques, it can be seen that in the poisoned model 1, which is the model with the weakest backdoor, they are not distinguishable from the other activations. In the other models we are able to detect them, but except from the poisoned model 2, which achieves a backdoor accuracy of about 90 %, the reduced activations are not clearly clustered into an own "group" different from the cluster with the normal activations.

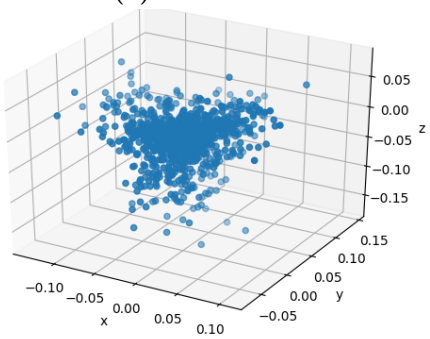
In spite of these not promising first impressions, we analyze the clusters' sizes, centroids' distances, and silhouette scores after reducing the dimension of the activations with both PCA, FastICA, 2-means, and the four types of agglomerative clustering. As centroids are only created with 2-means and not with agglomerative clustering, we restrict the evaluation to clusters' sizes and silhouette scores for the agglomerative clustering techniques. Furthermore, for the clusters' sizes, we always present just the proportion of data that is clustered into the smaller cluster. Table 15 presents an excerpt from our results. The other results can be found in Appendix A. All values are averaged over the values from the 5 clean and 5 poisoned models. As it can be seen, for the poisoned models the average clusters' sizes are smaller, and the centroids' distances as well as the silhouette scores are bigger than in the benign case. Hence, they seem to behave as we expected (cf. Section 4.4.2). However, the standard deviations of the poisoned models are significantly bigger than the standard deviations of the



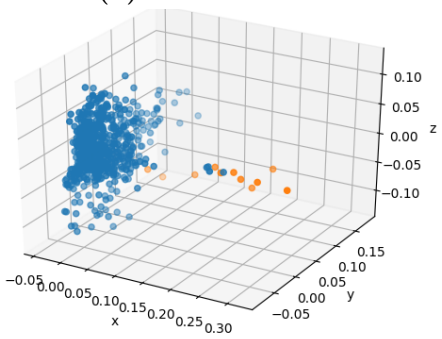
(a) Clean Model 1



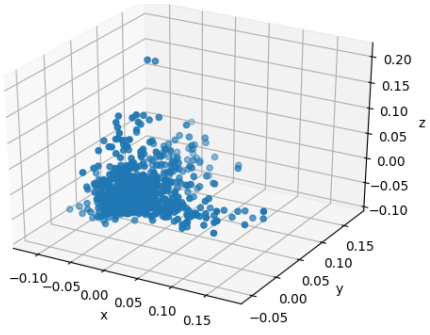
(b) Poisoned Model 1



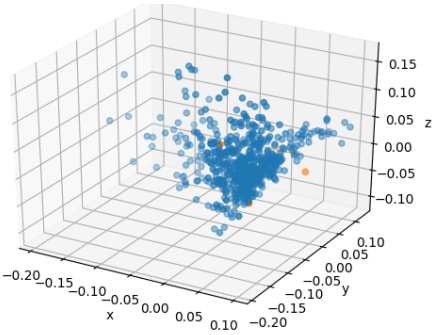
(c) Clean Model 2



(d) Poisoned Model 2



(e) Clean Model 3



(f) Poisoned Model 3

Figure 24.: CIFAR-10: Visualization of the reduced 3-dimensional activations after applying FastICA - Part 1

Dim.	Clusters' Sizes		Centroids' Distances		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**	**Clean**	**Poisoned**
3	0.2929105 (0.0658678)	0.19426 (0.1159221)	10.914418 (4.4644455)	13.3168739 (7.2822191)	0.3647463 (0.0623191)	0.4508000 (0.1548043)
4	0.29222210 (0.0664517)	0.1967 (0.1138779)	10.9284210 (4.4536989)	13.8090841 (8.5010333)	0.3261042 (0.0636190)	0.41606 (0.1726006)
10	0.2904263 (0.0648694)	0.19492 (0.1120568)	10.9569467 (4.4653188)	13.7170263 (8.1447425)	0.2422726 (0.0581941)	0.32694 (0.1620634)
15	0.2896326 (0.0652927)	0.18766 (0.0968921)	10.9723579 (4.4671899)	13.5462986 (7.7564899)	0.2167042 (0.0556342)	0.29368 (0.1560775)

Table 15.: CIFAR-10: Static experiments - Results for activation clustering with PCA and 2-means

benign models. This gives a hint why we achieve very poor detection results when we set a threshold for distinguishing clean and benign models between the clean and the poisoned averaged values for clusters' sizes/centroids' distances/silhouette score from the static experiments. If we applied such a threshold to an active learning process, we would test all classes of an updated model and check whether the value (e.g., the silhouette score) of one of its classes is above/under the threshold. In such a case, this class would be suspected to be poisoned. In the static experiments with the 10 models, we are able to observe that the proportion of how many poisoned classes are above such a threshold compared to benign classes is around 50 %, which is indistinguishable from random behavior. Additionally, we detect that nearly every model contains at least one class above the threshold in which case it would be marked as poisoned. Table 16 shows an example for such a threshold analysis with the silhouette score and reducing the activations to 3/15 dimensions. As it can be seen, we test several values as threshold that lie between the average clean and the averaged poisoned silhouette scores of the respective dimension in Table 15. For good true positive rates of at least 80 % the false positive rates are also at 80 % which is not acceptable for a good defense system.

Because of these not promising results, we decided not to proceed with active experiments. We were not able to find any distinguishing characteristics between clean and poisoned models with activation clustering which is why we discard this defense layer.

6.3. Feedback Loop

In our last set of experiments, we test the effect of a client-driven feedback loop that lets the clients analyze a new global model with their local test data. Thereby, any backdoor detection technique could be used by the clients. Based on the static results from the activation clustering presented in Section 6.2.1, we do not include that defense

Thresh.	3 Dim.	
	TP	**FP**
0.37	1	1
0.39	1	1
0.41	1	0.8
0.43	1	0.8
0.45	0.8	0.8
0.47	0.6	0.4
0.49	0.4	0.2

Thresh.	15 Dim.	
	TP	**FP**
0.21	1	1
0.23	1	1
0.25	0.8	1
0.27	0.8	0.8
0.29	0.8	0.8

Table 16.: CIFAR-10: Static experiments - Threshold analysis results for silhouette score with PCA & 2-means

layer into the feedback loop. Instead, we let the clients only analyze the class-specific misclassification distribution with respect to the two best performing aggregation rules for each metric from the active experiments. Thus, for Metric #1, we test ABS and SE for *target* classes, and for Metric #2 we use ABS and SUM for *target* classes as aggregation rules. Each client receives the last 10 global models together with the new updated global model that it is supposed to test. It determines the mean μ and standard deviation σ of the results for these metrics from the old models with its local test data and checks if the new update lies in the accepted range of 3σ around the calculated mean μ. If this is not the case, it will inform the server that it suspects the update to be poisoned. Otherwise, it trains based on the accepted model to create a new local update that it returns to the server.

As we observe that for Metric #2 a range of 3σ marks only very few of the poisoned updates as suspicious, we change the threshold range to 2σ. All results can be found in Table 17. Except from Metric #2 with aggregation rule SUM and 3σ, where we stop after 3 experiments, we average across 12 experiments. The best true positive rate of about 80 % is achieved with Metric #2 , SUM, 2σ, and the *target* classes. It contains an average false positive rate of approximately 9 %.

In a second set of experiments, we combine the feedback loop with the local testing of the server/aggregator as done in the active experiments in Section 6.1.2. Thereby, we choose Metric #1 with ABS, 3σ, and *target* classes for the server's analysis and Metric #2 with SUM, 2σ, and *target* classes for the clients' feedback loop, because they are the best performing combinations of metrics and aggregation rules in the previous experiments. The results shown in Table 18 are averaged across 10 experiments. As it can be seen, combining the two layers results in similar values of true positive rate compared to taking each layer on its own. Nevertheless, one has to take into account that here each layer has less data (25 %, 50 %, 75 %). Thus, the combination of the defenses can still be valuable in a federated learning setting, as there the server will

		TP	FP	Main Task Accuracy	Time (hours)
Metric #1	ABS, 3σ, Target	0.478611 (0.3746284)	0.0151234 (0.0149304)	0.7832	15.07
	SE, 3σ, Target	0.7186111 (0.3105077)	0.0254938 (0.0156958)	0.783	14.68
Metric #2	SUM, 3σ, Target	0.055 (0.0400689)	0.005 (0.0013096)	0.7812	16.37
	SUM, 2σ, Target	0.7963889 (0.0929914)	0.0824382 (0.0091604)	0.7780	14.58
	ABS, 2σ, Target	0.5594443 (0.2811800)	0.3081173 (0.1909801)	0.5382	12.45

Table 17.: CIFAR-10: Results of client-driven feedback loop

Proportion of data at Server/ Clients	TP	FP	% Server-TP	% Server-FP	Main Task Accuracy	Time (hours)
25/75	0.8416668 (0.1275433)	0.2236297 (0.274017)	0.92	0.54	0.7109	13.77
50/50	0.752 (0.173171)	0.1116296 (0.0091376)	0.84	0.45	0.7827	14.45
75/25	0.7896669 (0.1375217)	0.109778 (0.0116615)	0.94	0.51	0.78032	13.87

Table 18.: CIFAR-10: Results of a combination of local class-specific misclassification distribution analysis and client-driven feedback loop

probably just have a small data set compared to the data of all clients and the data is heavily distributed.

6.4. Defense Extension with TEEs

In our empirical evaluation, we focus on the basic attack proposed by Bagdasaryan et al. in [7]. The authors also introduce two strengthenings for the attack that aim to evade detection as mentioned in Chapter 3. The first one, called *Constrain-and-scale*, changes the objective function *loss* such that it does not only minimize the prediction error, but also incorporates an optimization for anomaly detection that an aggregator might apply to detect poisoned updates. The second enhancement, *Train-and-scale*, clips the returned update to an upper bound to evade the detection of a poisoning attempt by outliers' in local updates.

We do not evaluate our defense system on these adjustments, because they are supposed to evade detection methods that are orthogonal to ours. Nevertheless, we shortly discuss here an extension that is able to ward off both attack variants through the integration of Trusted Execution Environments (TEE) on the clients.

A TEE (as for example Intel SGX [1]) enables trusted computing via a combination of tamper-resistant hardware, secure storage, and software protection. The TEE provides isolation, ensuring that any data and code stored in the enclave's protected memory cannot be read or modified by the untrusted parts of the host system as for example the main operating system. Furthermore, it offers a remote attestation functionality, allowing the aggregating server to verify that the TEE only runs code known and approved by the server [92, 93, 94].

Figure 25 shows the setup for prohibiting the attack modifications of Bagdasaryan et al. in federated learning. In the beginning of the training process, every client provides a set of data that will be used for training and analysis to its TEE. With the hash of this data set the TEE can verify in later rounds that the data was not changed or complemented. This forces a malicious client to commit to its data in the beginning and reduces its possibility to adapt the injections. Before the first iteration, the server provides the objective function *loss* and all other training parameters, as for example the global learning rate η, via a secure communication channel directly to the TEE. The server can verify via remote attestation that a client is correctly running the class-specific misclassification distribution analysis, the training, and the determination of the local update on trusted hardware according to the provided parameters and objective function. The global and the local updates are directly communicated via secure channels between TEE and the aggregating server such that a malicious client has no access and cannot manipulate the objective function, training parameters, or apply clipping on the local update. Please note that due to the memory limitations of TEEs this protection might implicate performance reductions.

It follows that if we can detect the basic attack in an aggregated update, we are able to thwart the full-fledged Bagdasaryan et al.'s attacks with the TEE extension.

6.5. Discussion

Our experimental evaluation of the three defense layers reveals that the first layer (the class-specific misclassification distribution) and the third layer (the feedback loop), used solo and in concert, show promising results with detection rates around 80 % while keeping the false positives low enough to still reach the original (i.e., without the defense in place) main task accuracy of about 78 %. Unfortunately, the second layer

[1] `https://software.intel.com/en-us/sgx`

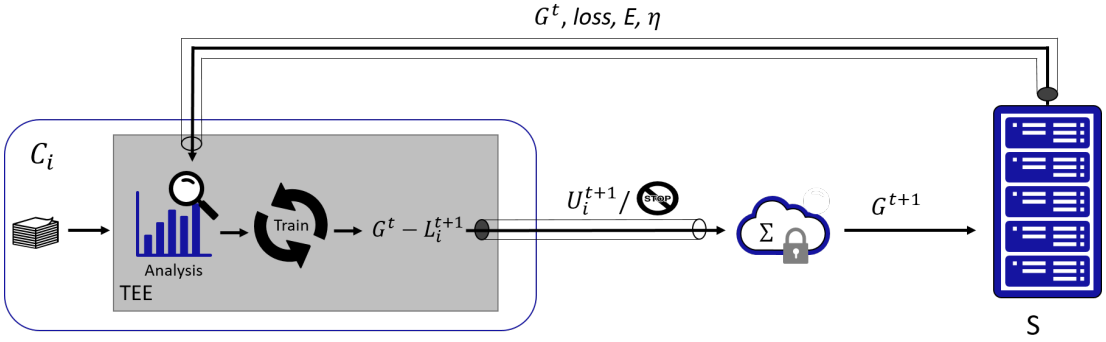


Figure 25.: System architecture using TEEs that can prevent attack enhancements

(the activation clustering) does not seem to be effective to detect poisoned updates in our setting with the tested techniques. This is surprising as prior art used a very similar approach to detect trojaned backdoors. Our expectation was that, from the algorithmic perspective of a neural network, it should not make a difference if the trigger/characteristic that causes the misclassification is artificially added or appears naturally in the data. Therefore, we expected the activation clustering analysis to highlight the presence of a backdoor. However, our experiments show no evidence for such a behavior, hence our intuition could not be validated.

With respect to the requirements defined in Section 4.3, we discuss our results for the two successful defense layers in this section.

6.5.1. Privacy

Federated learning was originally proposed to protect the privacy of clients' data when training machine learning models. Instead of sharing a client's personal data to a central party that trains the model, federated learning outsources the training procedure to the clients such that their data remains locally. This key characteristic of federated learning should also be preserved when defending against the injection of semantic backdoors. Furthermore, the individual model updates can leak some information about the training data as well, which is why it would be advantageous to apply a secure aggregation scheme that prevents the central server from accessing any individual local update and provides it only with the aggregated update for the global model.

When integrating the proposed defense system into the federated learning process, the data remains locally at the clients' devices (P-1) and it also allows to apply secure aggregation on the clients' local updates (P-2). Hence, it can be concluded that our two defensive layers do not degrade the privacy of the clients' data.

- **P-1 Privacy of User Data:** ☑

- **P-2 Secure Aggregation:** ☑

6.5.2. Security

A defense system that successfully protects federated learning against semantic backdoor injections should either detect any poisoning attempt, or it should at least ensure that the backdoor accuracy and durability achieved by the attacker remains low, so that the backdoor is ineffective.

Based on our experiments with the proposed defense systems, we are either able to detect the injection of a malicious contribution into an aggregated update (S-2), or the backdoor's effect is very modest (S-1) and short-term (S-3).

- **S-1 Weak Backdoor:** ☑
- **S-2 Detectable Backdoor:** ☑
- **S-3 Low Durability of the Backdoor:** ☑

6.5.3. Functionality

While a defense aims to thwart attack attempts on a system, it should nevertheless not significantly harm the original purpose of the main system. In the case of (federated) machine learning or, more specifically, neural networks, this means that the classification should still be done with high accuracy in an efficient manner in terms of computation and communication.

Our tests that integrated the two successful defense layers into the federated learning process show that with the best performing aggregation rule and metric the average main task accuracy (F-1) is about 78 %, meaning that it is not affected. Additionally, our defense system does not change the amount of messages that need to be sent between server and clients, because the analysis in the feedback loops is stacked in front of the training and done by the same clients. However, it increases the message's size significantly by a factor of 11. The static experiments described in Section 6.1.1 needed 16.08 hours on average, while the runtime of the active experiments with integrated defense system ranged from around 14 to 17 hours. Therefore, the training process is not significantly slower due to the defense system.

- **F-1 High Main Task Accuracy:** ☑
- **F-2 Low Computation Overhead:** ☑
- **F-3 Low Communication Overhead:** (☑)

7. Conclusion

In this thesis, we propose, implement, and test the first defense system against semantic backdooring attacks in federated learning while using secure aggregation to achieve a high level of data privacy for the clients. We intensively investigate prior work and find that existing backdoor attacks are not always as effective in terms of stealthiness and durability as expected. Moreover, we provide the first complete formal definitions for the three types of backdooring attacks. Furthermore, we formally define the investigated problem of backdooring federated learning, and the respective attacker objectives, before introducing the solution requirements and the threat model. Our experimental results show that two of the three layers form a promising direction for thwarting semantic backdoors in federated learning that fulfill all previously defined requirements, while the second layer could unfortunately not be validated to be a successful defense. As an extension, we demonstrate how more advanced detection evasion approaches can be impeded.

7.1. Limitations & Future Work

Due to time and hardware constraints this work leaves several aspects open for future investigation that we elaborate in the following.

Firstly, more configurations for our defenses can be experimentally evaluated. For instance, we fixed the number e of votes for the client-driven feedback loop required to reject an update to 3 to have a first test setting. However, it would be interesting to investigate the effect of different values for e. In addition, we tested our defensive layers with backdoor injections every 10th round, because this configuration resulted as the most successful from the attacker's perspective in our exploration of semantic backdoor attacks. Another extension would be to analyze how the defensive layers perform with other attack intervals. We focused on semantic backdoor attacks because it is the most serious backdoor attack type due to its good stealthiness and the convenient creation of backdoor samples. However, the effect of our two defensive layers on the two other backdoor attack types could be studied. Additionally, a more "democratic" combination of the two layers could be investigated where server and clients analyze a new update in parallel and both vote for accepting/rejecting the model. In such a scenario, one has

to study the optimal balance between the weights of the server's vote and the clients' votes.

Furthermore, we suggest to investigate further how clusters of activations evolve over time. So far we compared, for example, the static clusters' sizes of the reduced activations after each iteration and we could not find a boundary that directly distinguishes them. Another approach would be to study the development of the clusters over time. Similarly to the class-specific misclassification distributions, a backdoor injection might also cause some rapid changes in the clustering behavior and when comparing the clustering results with the last r rounds, poisoning could be detectable. Although we are not able to observe that poisoned samples create their own cluster, it is visible that after injection the clusters clearly look different. Therefore, a study of these developments could be valuable.

Additionally, we were unfortunately not able to extend our experiments to other data sets than CIFAR-10 due to the high expenditure of time that is required to test various configurations. As a next step, it would be especially interesting to test if our observations are also valid for a learning process with non-visual data, as for example a sentiment analysis or word prediction.

Another aspect in backdooring federated learning that is interesting to further investigate is how different data splittings (i.e., distributions of training data among the clients, as for example, IID, unbalanced, and various extents of truly non-IID) affect backdooring attacks. So far, we focused on an unbalanced data splitting following Bagdasaryan et al.'s work [7], and we already observed that unbalanced data supports the injection of backdoors. The open questions for the different types of data splittings are, for example, not only if they hide poisoning attempts by making malicious updates less visible through small variability in the main task accuracy and a long durability of the backdoor, but also if they support detection and defense by the aggregator and/or honest clients. For each data splitting, the accuracy and durability of the backdoor injection, and its effect on the main task accuracy and our two defense layers, should be analyzed. Artificial non-IID data could, for example, be created by giving only data from a certain amount of classes to each client (in an unbalanced way). A truly non-IID data set was created by Caldas et al. [91]. They sort data according to the creator. Thus, for instance all messages (and only these) from the same writer will belong to one client. Such a data distribution is interesting to simulate a realistic federated learning setting under attack as close as possible.

Furthermore, we assumed a static attacker so far. The natural next step would be to study our defense system under the attack of adaptive adversaries. Can adaptive attackers circumvent our defense system? If this is the case, and due to the privacy goals of federated learning, is it then possible at all to defend backdooring in federated

learning? What is the effect of having several malicious clients that collaborate or try to inject different backdoors?

To conclude, we make a first contribution to thwarting (semantic) backdoor attacks on federated learning through statistical means and a joint effort between honest clients and the central aggregating server, and point out several open questions for future research.

8. Acknowledgments

First and foremost, I would like to thank my thesis advisors Dr. Ghassan Karame and Dr. Giorgia Azzurra Marson of the security group at the NEC Laboratories Europe in Heidelberg for their extraordinary support and advice during my internship and the writing of my thesis. Dr. Karame gave highly valuable guidance to steer the work into the right direction whenever needed. Dr. Marson was always available to help with problems and to discuss about the ongoing research, experiments, and writing. Additionally, I am very grateful for their discussions exceeding the scope of the thesis helping me to find the right direction for my future career.

I would also like to express my sincere gratitude to my academic supervisor Prof. Melek Önen for her continuous support during my semester at EURECOM and afterwards. Her guidance and knowledge helped me during my first experiences with research and her advice on both research as well as on my career have been invaluable.

Bibliography

[1] J. Jordan, "Setting the learning rate of your neural network.." https://www.jeremyjordan.me/nn-learning-rate/. Accessed: 2019-04-12.

[2] A. Sharma, "What is the differences between artificial neural network (computer science) and biological neural network?." https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network. Accessed: 2019-04-09.

[3] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *ACM Conference on Computer and Communications Security*, pp. 1175–1191, ACM, 2017.

[5] A. Karpathy, "Lessons learned from manually classifying cifar-10." http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/, 27.04. 2011. Accessed: 2019-07-04.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, vol. 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, PMLR, 2017.

[7] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *CoRR*, vol. abs/1807.00459, 2018.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[10] B. Widrow and M. E. Hoff, "Neurocomputing: Foundations of research," ch. Adaptive Switching Circuits, pp. 123–134, Cambridge, MA, USA: MIT Press, 1988.

[11] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, p. 1, Feb 2015.

[12] P. Russom, "Big data analytics," 2011.

[13] Domo, "Data never sleeps 6.0.." `https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf`. Accessed: 2019-07-15.

[14] J. Sun and C. K. Reddy, "Big data analytics for healthcare," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, (New York, NY, USA), pp. 1525–1525, ACM, 2013.

[15] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2561 – 2573, 2014. Special Issue on Perspectives on Parallel and Distributed Processing.

[16] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology – ICISC 2009* (D. Lee and S. Hong, eds.), (Berlin, Heidelberg), pp. 229–244, Springer Berlin Heidelberg, 2010.

[17] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 206–215, ACM, 2003.

[18] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, (New York, NY, USA), pp. 439–450, ACM, 2000.

[19] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, pp. 571–588, Oct. 2002.

[20] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber, "Privacy: Theory meets practice on the map," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, (Washington, DC, USA), pp. 277–286, IEEE Computer Society, 2008.

[21] L. Sweeney, "Only you, your doctor, and many others may know," in *Technology Sciene*, 2015.

[22] M. Arrington, "Aol proudly releases massive amounts of private data." https://techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data/?guccounter=1, 2006. Accessed: 2019-07-15.

[23] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, pp. 201–210, 2016.

[24] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving user clustering in a social network," in *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 96–100, IEEE, 2009.

[25] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo, "Analyzing federated learning through an adversarial lens," *International Conference on Machine Learning (ICML)*, vol. 97, pp. 634–643, 2019.

[26] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Magazine*, vol. 10, pp. 8–39, Jan 1993.

[27] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, (San Francisco, CA, USA), pp. 1137–1143, Morgan Kaufmann Publishers Inc., 1995.

[28] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*, 2012.

[29] L. N. Smith, "Cyclical learning rates for training neural networks," *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, 2017.

[30] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *International Conference on Learning Representations (ICLR 2017)*, 2017.

[31] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[32] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4520–4524, IEEE, 2015.

[33] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[34] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84 – 96, 2014.

[35] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[36] A. Kurenkov, "A 'brief' histroy of neural nets and deep learning." `http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/`. Accessed: 2019-04-10.

[37] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[38] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[39] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[40] C. W. A. H. J. K. John McGonagle, George Shaikouski, "Backpropagation." `https://brilliant.org/wiki/backpropagation/`. Accessed: 2019-04-24.

[41] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*, pp. 420–434, Springer, 2001.

[42] P. M. Domingos, "A few useful things to know about machine learning.," *Commun. acm*, vol. 55, no. 10, pp. 78–87, 2012.

[43] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014.

[44] Z. Jaadi, "A step by step explanation of principal component analysis," 2019.

[45] A. Hyvärinen, "Fast and robust fixed-point algorithms for independent component analysis," *IEEE TRANS. NEURAL NETW*, vol. 10, no. 3, pp. 626–634, 1999.

[46] P. Comon, "Independent component analysis, a new concept?," *Signal processing*, vol. 36, no. 3, pp. 287–314, 1994.

[47] C. Jutten and J. Herault, "Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture," *Signal processing*, vol. 24, no. 1, pp. 1–10, 1991.

[48] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[49] A. K. Jain, R. C. Dubes, *et al.*, *Algorithms for clustering data*, vol. 6. Prentice hall Englewood Cliffs, NJ, 1988.

[50] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*. Wiley Publishing, 5th ed., 2011.

[51] J. H. W. Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[52] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987.

[53] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[54] O. Fercoq, Z. Qu, P. Richtárik, and M. Takáč, "Fast distributed coordinate descent for non-strongly convex losses," *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 05 2014.

[55] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 850–857, Sep. 2014.

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

[57] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), pp. 1322–1333, ACM, 2015.

[58] T. Orekondy, S. J. Oh, B. Schiele, and M. Fritz, "Understanding and controlling user linkability in decentralized learning," *CoRR*, vol. abs/1805.05838, 2018.

[59] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, May 2017.

[60] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.

[61] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, pp. 644–654, Sept. 2006.

[62] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 119–129, Curran Associates, Inc., 2017.

[63] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," *arXiv e-prints*, p. arXiv:1206.6389, Jun 2012.

[64] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 3517–3529, Curran Associates, Inc., 2017.

[65] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, *et al.*, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural networks: the statistical mechanics perspective*, vol. 261, p. 276, 1995.

[66] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[67] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[68] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*, The Internet Society, 2018.

[69] T. J. L. Tan and R. Shokri, "Bypassing backdoor detection algorithms in deep learning," *CoRR*, vol. abs/1905.13409, 2019.

[70] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[71] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[72] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *CoRR*, vol. abs/1808.04866, 2018.

[73] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[74] C. Xie, S. Koyejo, and I. Gupta, "Practical distributed learning: Secure machine learning with communication-efficient local updates," *CoRR*, vol. abs/1903.06996, 2019.

[75] S. Shen, S. Tople, and P. Saxena, "Auror: defending against poisoning attacks in collaborative deep learning systems," in *ACSAC*, pp. 508–519, ACM, 2016.

[76] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A defence against trojan attacks on deep neural networks," *CoRR*, vol. abs/1902.06531, 2019.

[77] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," in *SafeAI@AAAI*, vol. 2301 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019.

[78] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 1484–1497, Dec 2012.

[79] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 115–124, Association for Computational Linguistics, 2005.

[80] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *S&P 2019*, 2019.

[81] L. Wolf, T. Hassner, and I. Maoz, *Face recognition in unconstrained videos with matched background similarity*. IEEE, 2011.

[82] "Attribute and Simile Classifiers for Face Verification," in *IEEE International Conference on Computer Vision (ICCV)*, Oct 2009.

[83] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Research in Attacks, Intrusions, and Defenses* (M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, eds.), (Cham), pp. 273–294, Springer International Publishing, 2018.

[84] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[85] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR*, vol. abs/1712.07557, 2017.

[86] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *CoRR*, vol. abs/1811.06965, 2018.

[87] D. Mishkin and J. Matas, "All you need is a good init," in *ICLR*, 2016.

[88] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[89] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *CoRR*, vol. abs/1806.00582, 2018.

[90] T. Minka, "Estimating a dirichlet distribution," 2000.

[91] S. Caldas, P. Wu, T. Li, J. Konecný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," *CoRR*, vol. abs/1812.01097, 2018.

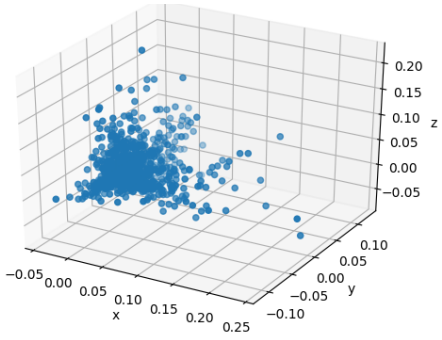
[92] D. B. S. G. Ben A. Fisch, Dhinakaran Vinayagamurthy, "Iron: Functional encryption using intel sgx." Cryptology ePrint Archive, Report 2016/1071, 2016. `https://eprint.iacr.org/2016/1071`.

[93] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019.

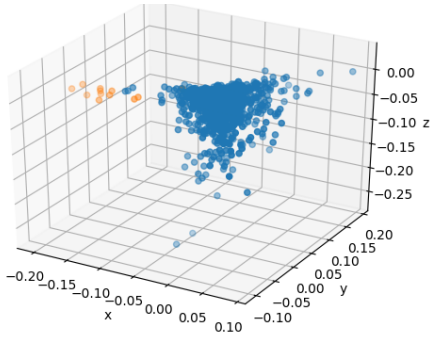
[94] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 57–64, IEEE, 2015.

Appendices

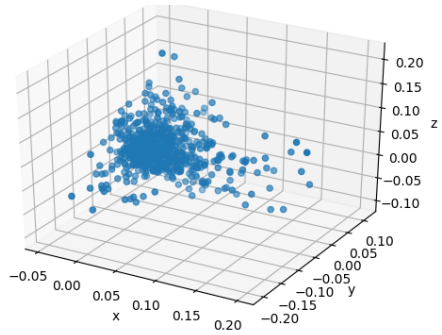
A. Additional Results for Defense Layer #2 (Activation Clustering)



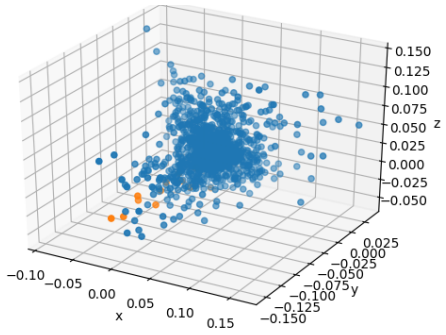
(a) Clean Model 4



(b) Poisoned Model 4

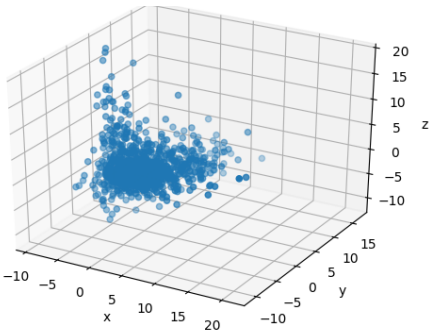


(c) Clean Model 5

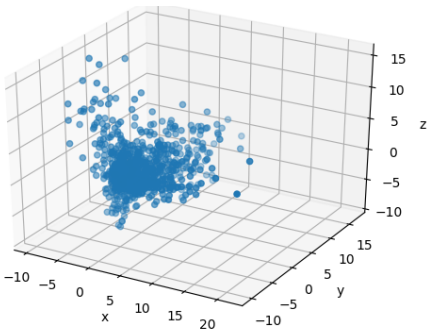


(d) Poisoned Model 5

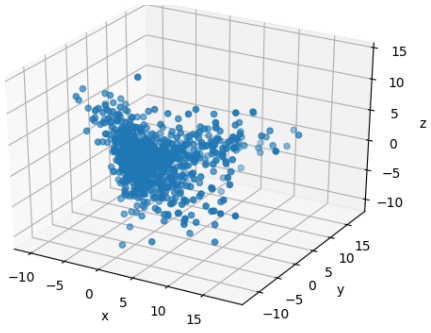
Figure 26.: CIFAR-10: Visualization of the reduced 3-dimensional activations after applying FastICA - Part 2



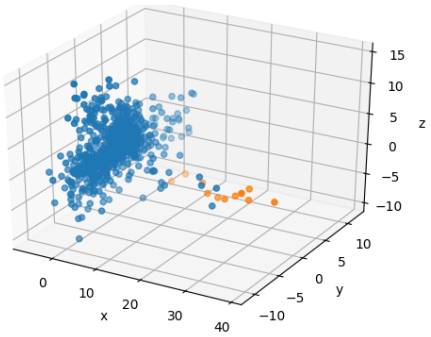
(a) Clean Model 1



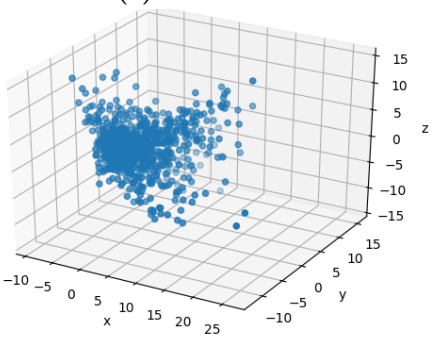
(b) Poisoned Model 1



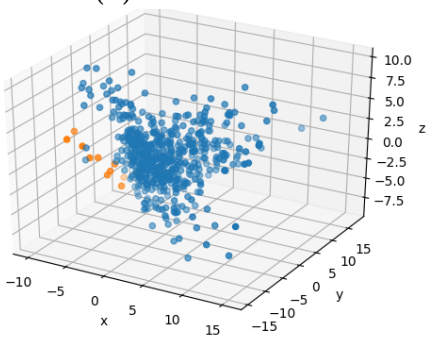
(c) Clean Model 2



(d) Poisoned Model 2



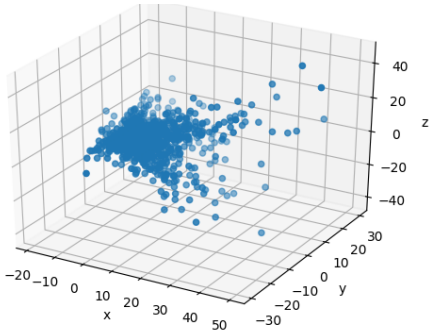
(e) Clean Model 3



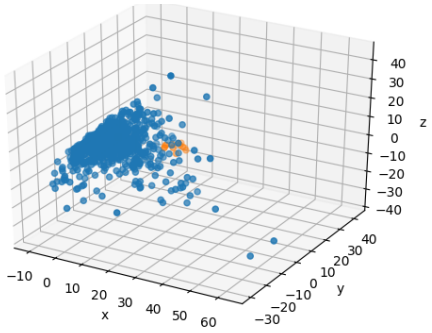
(f) Poisoned Model 3

Figure 27.: CIFAR-10: Visualization of the reduced 3-dimensional activations after applying PCA - Part 1

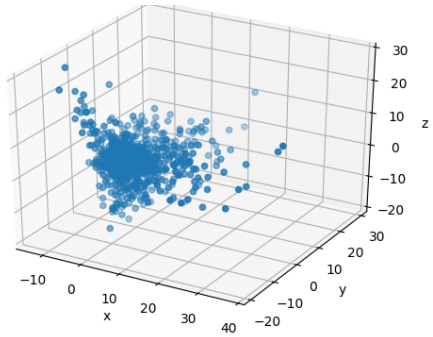
.



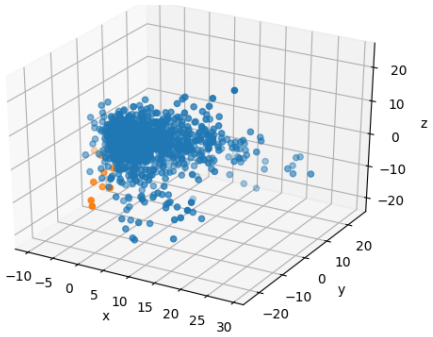
(a) Clean Model 4



(b) Poisoned Model 4



(c) Clean Model 5



(d) Poisoned Model 5

Figure 28.: CIFAR-10: Visualization of the reduced 3-dimensional activations after applying PCA - Part 2

.

Dim.	Clusters' Sizes		Centroids' Distances		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**	**Clean**	**Poisoned**
3	0.2754957 (0.0652617)	0.18118 (0.0927287)	0.0616025 (0.0126756)	0.0929746 (0.0605091)	0.3177789 (0.0446779)	0.41914 (0.139968)
4	0.2769189 (0.0656036)	0.18168 (0.0916401)	0.0618914 (0.0124578)	0.0987149 (0.0726818)	0.2663884 (0.0446631)	0.3702 (0.1487678)
10	0.2731831 (0.0748290)	0.21112 (0.0837238)	0.0638970 (0.0172905)	0.0737023 (0.0095202)	0.1501621 (0.0435908)	0.18548 (0.0285709)
15	0.2779010 (0.0805860)	0.1203 (0.0894316)	0.0639183 (0.0153342)	0.1307446 (0.0882444)	0.1185473 (0.0371158)	0.25338 (0.0715093)

Table 19.: CIFAR-10: Static experiments - Results for activation clustering with FastICA and 2-means

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.0012789 (0.0007108)	0.00532 (0.0075404)	0.6074516 (0.0678741)	0.68474 (0.0869073)
15	0.0012789 (0.0007108)	0.00532 (0.0075404)	0.5122547 (0.0773939)	0.6319 (0.1080505)

Table 20.: CIFAR-10: Static experiments - Results for activation clustering with PCA and agglo. clustering with single linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.0012347 (0.0006787)	0.00532 (0.0075404)	0.5986842 (0.0671347)	0.66828 (0.0817767)
15	0.0011379 (0.0006323)	0.0012 (0.0004301)	0.4897263 (0.0935056)	0.63268 (0.1014555)

Table 21.: CIFAR-10: Static experiments - Results for activation clustering with FastICA and agglo. clustering with single linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.1079516 (0.099599)	0.07384 (0.1382181)	0.4080979 (0.1173263)	0.58472 (0.1998557)
15	0.0560537 (0.0695815)	0.00682 (0.0114742)	0.37182737 (0.1382363)	0.622 (0.1002464)

Table 22.: CIFAR-10: Static experiments - Results for activation clustering with PCA and agglo. clustering with complete linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.1126147 (0.1128011)	0.01906 (0.0121362)	0.3714084 (0.1335924)	0.5664 (0.0826422)
15	0.0489074 (0.1023601)	0.00552 (0.0064263)	0.3559242 (0.1822069)	0.57164 (0.1767635)

Table 23.: CIFAR-10: Static experiments - Results for activation clustering with FastICA and agglo. clustering with complete linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.01485368 (0.0230617)	0.00566 (0.0073961)	0.5567653 (0.073659)	0.68722 (0.0779684)
15	0.0046053 (0.0077434)	0.00466 (0.0078596)	0.5042326 (0.0856276)	0.65688 (0.0636293)

Table 24.: CIFAR-10: Static experiments - Results for activation clustering with PCA and agglo. clustering with average linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.0063589 (0.0079168)	0.00814 (0.0070695)	0.57018 (0.0729454)	0.65372 (0.0728416)
15	0.0015263 (0.0010564)	0.00164 (0.0007765)	0.4921632 (0.0874293)	0.63724 (0.0981343)

Table 25.: CIFAR-10: Static experiments - Results for activation clustering with FastICA and agglo. clustering with average linkage-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.2759557 (0.1282749)	0.18502 (0.1339706)	0.3400663 (0.0858518)	0.44366 (0.1783744)
15	0.25878 (0.1106485)	0.16214 (0.1187286)	0.19920526 (0.0703863)	0.2655 (0.1842889)

Table 26.: CIFAR-10: Static experiments - Results for activation clustering with PCA and agglo. clustering with Ward-criterium

Dim.	Clusters' Sizes		Silhouette Score	
	Clean	**Poisoned**	**Clean**	**Poisoned**
3	0.2681105 (0.1088028)	0.14374 (0.1047018)	0.2812936 (0.0573381)	0.44152 (0.1712050)
15	0.1625252 (0.0935566)	0.03636 (0.0339217)	0.1207136 (0.0568604)	0.28162 (0.1171896)

Table 27.: CIFAR-10: Static experiments - Results for activation clustering with FastICA and agglo. clustering with Ward-criterium