FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Federated Machine Learning

Evaluierung von Backdoor-Attacken in neuronalen Netzwerken zur Bildklassifikation

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Business Informatics

eingereicht von

Florian Nuding, BSc.
Matrikelnummer 01425734

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Mitwirkung: Mag. Dipl.-Ing. Rudolf Mayer

Wien, 6. Mai 2020

Florian Nuding	Andreas Rauber

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Federated Machine Learning

An evaluation of backdoor attacks on image classification data

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Business Informatics

by

Florian Nuding, BSc.
Registration Number 01425734

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Assistance: Mag. Dipl.-Ing. Rudolf Mayer

Vienna, 6th May, 2020

Florian Nuding	Andreas Rauber

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Florian Nuding, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Mai 2020

Florian Nuding

Kurzfassung

Verteiltes maschinelles Lernen, auch bekannt als kollaboratives Lernen, hat sich neuerdings als effektives Werkzeug für die Verarbeitung von Daten, die an unterschiedlichen Orten vorliegen oder produziert werden, empfohlen. Ein grundlegender Vorteil dieser Methodik ist, dass diese verteilten Daten nicht erst zentralisiert werden müssen, sondern direkt an der Quelle verarbeitet werden können. Dies ist mit zahlreichen Vorteilen verbunden. Unter anderem, dass durch die direkte Verarbeitung die Notwendigkeit eines leistungsstarken, zentralen Servers nicht gegeben ist, was mit Einsparungen von Ressourcen und damit Kosten verbunden ist. Weiters können die Daten bei den Teilnehmern verbleiben und müssen nicht durch potentiell unsichere Netzwerke an potentiell unsichere Server übertragen werden, was einen großen Vorteil hinsichtlich Datenschutz bietet.

Doch durch die verteilte Funktionsweise dieser Technologie entstehen auch neue Angriffspunkte, die sich Aggressoren zu Nutze machen können. Angriffe auf maschinelles Lernen werden unter dem Begriff „feindliches maschinelles Lernen“ zusammengefasst, und können mithilfe des CIA-Dreiecks in Attacken auf die Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit kategorisiert werden. Der Fokus dieser Arbeit liegt auf Attacken hinsichtlich der Integrität in der Trainingsphase des Modells, genauer gesagt so genannten „Backdoor Attacken“. Dabei wird ein wiederholendes Muster in die Trainingsdaten eingefügt. Das Ziel dabei ist, dass in der Test-Phase des Machine-Learning-Modells ein feindliches Verhalten hervorgerufen wird – beispielsweise gezielte Missklassifikation der Daten.

In dieser Arbeit untersuchen wir verteiltes maschinelles Lernen auf Unterschiede zu zentralisiertem maschinellem Lernen hinsichtlich Effektivität der resultierenden Modelle. Außerdem werden Machbarkeit und Effektivität von Backdoor Attacken geprüft. Dazu werden Case Studies auf State of the Art Datensätzen durchgeführt und unsere Ergebnisse auf bereits vorliegenden Resultaten der Literatur evaluiert. Wir zeigen, dass verteiltes maschinelles Lernen eine vergleichbare Performance zu zentralem maschinellem Lernen bietet, gleichzeitig aber sehr verwundbar gegenüber Backdoor Attacken ist.

Abstract

Federated Machine Learning, sometimes also referred as collaborative learning, has recently awakened interest as a concept to process data distributed across many individual sources without the need to centralize it. The main idea behind this is that clients train models, based on their own data locally and only publish the parameters of the models. These parameters are aggregated into a global model which is subsequently shared across all participants.

The usage of federated learning strategies enables privacy, especially relevant when processing sensitive data, as the data itself is never shared across the network. Furthermore, applications can benefit from the advantage of the distributed structure by utilizing the computational resources on the clients' endpoints.

Adversarial Machine Learning describes a collection of techniques with a common goal of attacking artificial learning systems in respect to their confidentiality, integrity or availability. Recent research has shown that beside the above mentioned advantages, federated learning also enables new possibilities and entry points for adversaries due to its distributed nature. This thesis has its focus on backdoor attacks, a strategy interfering with the model's integrity during the training phase. By altering certain inputs with a reoccurring pattern during model training this attack tries to trigger malicious behaviour in the deployment phase.

In this work, we focus on evaluating the performance of different federated learning architectures. Moreover, we study the impact of backdoor attacks on image datasets in the domains of traffic sign classification and facial recognition.

Extending earlier work, we also include the setting of sequential (incremental cyclic) learning in our investigations and perform an in-depth analysis on several hyper-parameters of the adversaries. We show that federated learning performs on a similar niveau as centralized machine learning, but it is indeed vulnerable to backdoor attacks.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The volume of data produced in the world is drastically increasing and likely to nearly double its volume in three years [RGR18]. Machine learning tries to extract value out of this large amount of data by using different strategies, like creating classification models to gain a deeper insight on the data. The training process of these models becomes more complex when data volume increases and requires lots of computational power. Until now, the de facto standard of processing big data sets was moving them into the cloud or to a data cluster, where a lot of resources are available. But with a growing number of endpoints and potentially useful computational capacities, as well as the rise of data privacy awareness and new data protection laws, especially the General Data Protection Regulation in the European Union, the demand for alternatives to centralized computation is growing.

A possible solution to these privacy related issues is Federated Learning. Rather than moving the data to the model, the approach of Federated Learning [KMRR16] is based on the principle of creating the model where the data is generated. Afterwards, these models are combined into a global model, e.g. through a central server. Its main advantages are data itself being never sent over the network but kept locally on the users' endpoints, while the model generation is outsourced to the endpoints [KMY+16]. Furthermore, this also enables model training on heterogeneous data [YLCT19].

A typical use case to Federated Learning is depicted in the following example. Imagine someone visits a hospital as she does not feel healthy. Based on results of medical tests like blood pressure, pulse or blood screening results, aswell as the doctor's experience, the doctor is able to deliver a targeted diagnosis on the person's health condition. However,

this diagnostic process can also be digitized and translated into some machine learning model, for example a decision tree, to support doctors and preserve experience through generations (e.g. in [JH19]). As the number of different diseases is very large, it is unlikely that each doctor has experience in linking several syndromes to every possible disease. Doctors could share data about their patients to share their experience. However, sharing sensitive patient data, especially throughout doctors working in different hospitals can lead to big privacy issues.

The usage of federated learning addresses this issue as sensitive data is not directly sent over a potentially insecure network, but only parameter updates from locally trained models. This allows participants of the federated network to benefit from other participants' data without explicit sharing of training inputs [SS15].

1.2 Problem Statement

Beside the above mentioned advantages federated machine learning offers, several serious attacks exist and try to exploit the model. Outsourcing the training process to potentially insecure or malicious clients creates even more entry points for attackers. An overview on different attacks can be given by looking at the famous CIA triangle. This triangle consists of three dimensions, namely confidentiality, integrity and availability, and represents a foundation in IT security. Every security vulnerability can be viewed by one or a combination these three concepts.

Each of these dimension can be attacked in a federated learning setting. According to literature [GDG17], especially "Adversarial Inputs" and "Backdoor attacks" have a high potential in damaging the intended use case of a machine learning model .

Adversarial Inputs describes a range of attacks targeting an already trained machine learning model. An attacker tries to add some specific, but hardly recognizable changes to some input data, with the goal of producing a misclassification of the input.

Backdoor attacks, in contrast, try to poison the training data during the model's learning process. By inserting wrongly labelled data, an attacker tries to embed a specific pattern into the model to trigger malicious behaviour. In an optimal case, the model's performance on unaltered data should remain the same, while as soon as the model gets an altered input containing this exact pattern, the malicious behavior is executed. These attacks are explained in more detail in Section 2.5.

1.3 Research questions

The aim of this work is an evaluation of different federated learning approaches, their advantages and disadvantages in comparison to central machine learning, as well an evaluation on potential systematic weaknesses.

Furthermore, we address potential weaknesses of federated learning by evaluating the impact of backdoor attacks on the model. In these integrity based attacks, an adversary

enters certain patterns into the data during the training phase to trigger malicious behaviour. A more detailed description of these attacks is given in Section 2.6.

Broken down into the following topics, we investigate the influence of certain parameters measured by the common effectiveness metrics like accuracy:

1.3.1 Number of clients

First, an emphasis is put on the number of clients participating in a machine learning network. While centralized machine learning works using one client processing all data, federated learning usually consists of data being distributed among dozens of clients. We investigate if a bigger number of participants in a network results in the same model performance as a smaller number?

Relevance:
This research question is especially relevant as the fundamental idea of federated machine learning is processing data in a decentralized way. Optimally, having the data distributed across many sources comes without the drawback of lowered model effectiveness.

Methodology and Measurement of success:
We investigate prior work and experiment on benchmark data at different numbers of clients to draw conclusions on the behaviour. Success is measured by comparing the accuracy on models trained on different numbers of clients, and evaluating results against literature.

1.3.2 Distribution of the data

A previously mentioned advantage of federated learning, the unnecessity of transferring all data to a central machine learning instance, can lead to non independent and non identically distributed (non-iid) data. We investigate if sparsely known data classes are learned by the same level as more commonly known classes, in comparison the same model being trained in a centralized setting?

Relevance:
Processing non-iid data is of high relevance in a federated learning setting, as it is very likely that not all classes are known by each client but certain data classes are only to known certain clients.

Methodology and Measurement of success:
We measure success in the per class effectiveness of classes only known by some clients, in respect to classes known by all clients. Again, we compare our findings to behaviour on non-iid data by prior work.

1.3.3 Backdoor attacks and the pattern's prominence

As mentioned, backdoor attacks require a certain pattern added to data used in training of the machine learning model. These patterns can vary in size and shape. We investigate

to which extend one can introduce malicious behaviour into a machine learning model while adding backdoors that are noticeable but not overly suspicious?

Relevance:
Machine learning models have been successfully attacked using backdoor attacks in the past (see Section 2.6), and the most fundamental property of this attack is the backdoor pattern itself.

Methodology and Measurement of success:
The rate of success is measured, on the one hand, by the level of accuracy a set of poisoned data is successfully misclassifiied to the targeted class, while the performance on non-backdoored data simultaneously on above a threshold. Exact values depend on the data's use-case.

On the other hand, patterns are, due to the structure of backdoor attack, clearly visible, but as a goal they should be as unsuspicious (in respect to the data itself) as possible. The suspiciousness of a pattern is not quantifiable and also depends on the domain of the data. We evaluate these attacks on two different benchmark datasets in common areas of image classification.

1.3.4 Backdoor attack strategies

In the setting of federated learning, an attacker can use different attack strategies to introduce a backdoor into the global model. In a basic strategy, an attacker poisons an amount of training data, trains the model locally and shares it with other clients. Literature (see Section 2.6.1) also describes more complex methods. We investigate two state of the art attacks strategies. The goal is to determine which strategy is more suitable for an adversary to use, in respect to effectively introduce the backdoor into the machine learning model while preserving effectiveness on benign data?

Relevance:
The relevance of this question is given as advanced attack strategies might be able to introduce malicious behaviour faster or to a higher degree into the model, a fundamental goal of backdoor attacks.

Methodology and Measurement of success:
Success is measured by comparing accuracies of both methods on benign and poisoned data, by analysing observations achieved by prior work as well as our experimental results.

1.3.5 Number of adversaries

We believe that an important factor to the success of the attack is the number of attackers in a federated machine learning network. Investigations are performed on the number of benign clients in relation to clients containing malicious data. We want to investigate to what extend the ratio of benign to fraudulent clients has an impact on the attack?

Relevance:
It is of valuable information for an attacker to gain insight on the number of clients he needs to control to make the attack work.

Methodology and Measurement of success:
We measure success by again looking the accuracy of the main and backdoor task on a varying numbers of clients, and compare our results to prior work.

1.3.6 Point of time backdoors are inserted

Finally, in a certain type of federated learning, the so called sequential learning, we believe that the order of backdoor insertion has a significant impact on the performance of the backdoor in the global model. In this specific type of federated learning, the clients' models are not averaged, but they are combined sequentially and iteratively one-by-one. For the adversary it is of high importance to know at which point of the learning cycle he should inject the poisoned data? As an hypothesis, we believe that the later the poisoned data is processed, the more successful the backdoor will be introduced into the global model due to potential memorizing effects.

Relevance:
An adversary could use this knowledge to strengthen his attack by interfering at an optimal point of time, while for a defender it might help in securing the federated learning process more effectively.

Methodology and Measurement of success:
We experiment with different sized sequential learning networks on benchmark datasets, and vary the point of time the model is trained on malicious data. Using potential differences in effectiveness on the main and backdoor task we try to obtain insight on the behaviour of sequential learning.

1.4 General methodological approach

All insights are achieved by using a methodology called "design science" proposed by Hevner et. al [HRM+04]. After reviewing literature related to our topics, according to the first step of this approach, we design an artifact in our defined problem domain. This artifact is provided in form of a case study on benchmark datasets to investigate our research questions formulated above. The second step consists of an evaluation of our results, which is in our case obtained by comparing our observations previous insights gained from literature. A more detailed explanation on the methodology is given in Chapter 3.

CHAPTER 2

Related work

This section gives an overview on the related work and the state of the art.

2.1 Machine Learning

Machine Learning is the process of artificially generate knowledge based on experience. Algorithms learn from data to make predictions automatically, without human's assistance to specify rules. The field of machine learning can generally be divided into *Supervised Learning* and *Unsupervised Learning*.

In supervised learning, one trains the machine with labeled data - meaning that the data is correctly tagged and representing the ground truth. Based on this labeled data, the machine is able to predict non-labeled, unseen data. Types of supervised learning are classification and regression.

In contrast, unsupervised learning does not use labeled data. It rather discovers information on data its own, like finding hidden patterns or features.

Our work has its focus on supervised learning, specifically the task of classifying data.

2.2 Classification task & problem setting

Classification describes the process of determining the target or label of a certain data point. For example, classification enables one to decide whether a patient is ill or not, based on the data one knows about him. Algorithms which enable classification can in general be divided into two categories: Lazy Learners and Eager Learners.

Lazy learners just store data and start the classification process when they receive test data. The data-storing step is very fast, but as there is no training-like process involved, the classification of "new" data points can take a lot of time. Each training data point

has to be processed in order to correctly classify "new" data. An example for a lazy learner is the "k-nearest neighbors (KNN)" algorithm.

Eager learners follow a different approach. When they first receive data they start to build a model on this so called "training data", enabling classification for "new" data. In contrast to lazy learners, they do not wait for test-observations as to start processing. From a run-time perspective, this training phase normally uses way more time than the testing phase. Famous eager learners are decision trees, support vector machines or Bayesian classification. This eager machine learning process is depicted in Figure 2.1.

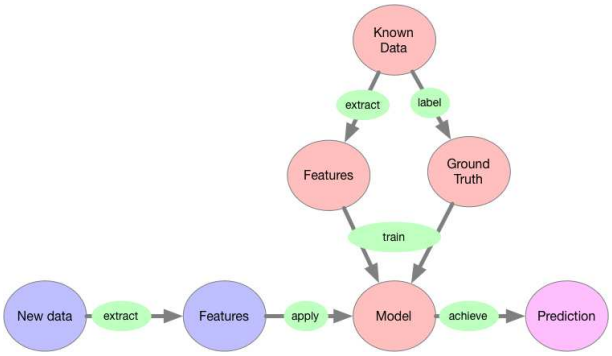


Figure 2.1: A diagram of the machine learning process

This work mainly focuses on the classification of images. Non-image datasets often consist of many observations, each represented by explicitly expressed features. In the case of patient data possible features are blood pressure, body temperature or gender. Images, however, do not have such explicit properties. When using the above described algorithms, the images' features, for example edges, shapes or certain other portions of interests have to be extracted first. Also, the features' location is of importance. To solve this problem one typically uses techniques for feature extraction and detection in combination.

For feature extraction, an algorithm called "Principal Component Analysis (PCA)" is often used. The goal of PCA is to reduce the dimensionality of the data and extract a smaller number of features that still represent a significant part of the information. For feature detection, one can use computer vision technologies like "Histogram of oriented gradients" or "Local binary patterns".

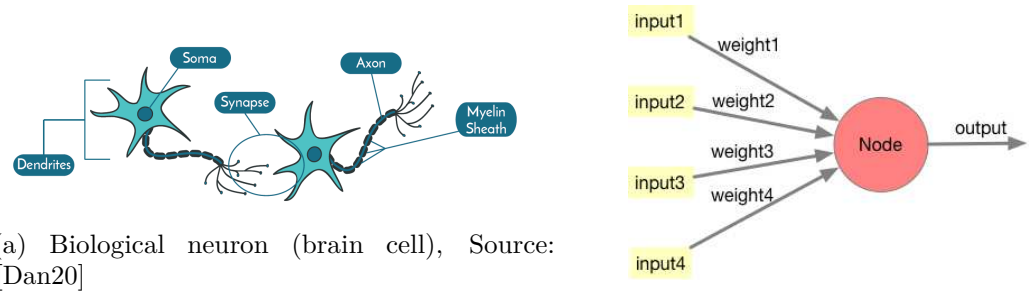
A more recent approach for targeting images is the usage of deep learning technologies. They have the advantage that it is unnecessary to extract the images' features manually. Rather they implicitly learn their features during model training. The only thing one needs in advance is a labelled dataset. A very common set of technologies in the field of deep learning are neural networks.

2.3 Neural Networks

A neural network is a machine learning method inspired by biology and the biological neural networks e.g. in the human brain. Neural networks follow the principle of trying to generate knowledge based on experience, but generally without a need of being programmed with task-specific rules. To achieve this, an (artificial) neural network consists of one or more neurons, that are connected to each other by "weights". A neuron receives numeric values as input, and forwards them to the next neuron by multiplying its own value by its outgoing weight. By changing this weight in a specific manner we can make the model fit our needs.

One can create the some analogies on how biological and artificial neurons are built and work. In a brain cell, dendrites make up the basis for perception of signals. When perceived, they are processed in the cell nucleus. When enough signals are received, they are sent over the axon. There, the synapses are connected, which permit neurons to pass signals to another neurons.

Similarly, the artificial neurons' inputs are connected to the node via weights, processed in the node and sent over to other neurons via the output. A visual comparison is depicted in Figure 2.2.



(a) Biological neuron (brain cell), Source: [Dan20]

(b) Artificial neuron (Perceptron)

Figure 2.2: Comparison of biological and artificial neuron

There exist several different neural network architectures, but the simplest is the "Perceptron" [Ros58], depicted in Figure 2.2b. This structure consists of only one neuron, which takes n input values, multiplies each of them with its own weight vector, and delivers one single output value.

A more complex network is the so called "Multilayer Perceptron (MLP)", shown in Figure 2.3. This architecture consists of at least three layers: an input layer, one or more hidden layers and an output layer. The input layer contains the data to be processed in the MLP in a numerical shape. These hidden layer(s) consist of several Perceptrons ordered in parallel and connected in series. They are connected by weights and biases (for shifting), and after each Perceptron an activation function follows. This function is applied after building the sum of all input values multiplied by their corresponding weight. It is necessary to determine whether a neuron's input is relevant for predicting the model,

in other words if a neuron should be activated or not. It also helps normalizing the output of the neurons, usually between 0 and 1. MLPs are only used for very simple classification tasks.

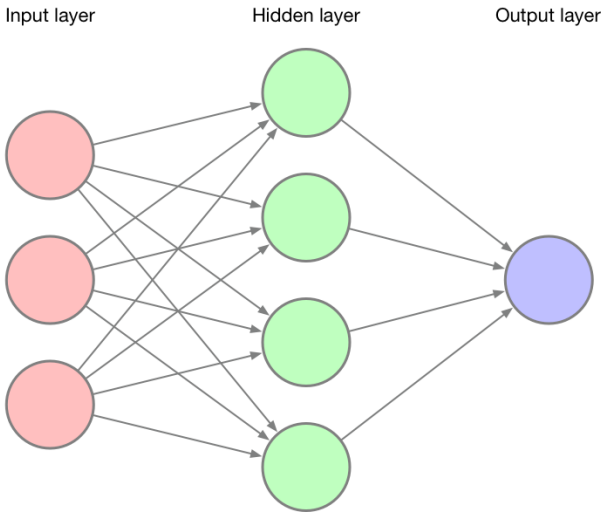


Figure 2.3: A diagram of a Multilayer Perceptron

In a neural network, the current state of a model is used to give a prediction on the labels of a *batch* of training data. These predictions are then compared to the correct labels, the ground truth, of the training data. The resulting difference is a statistical estimator of the error gradient, which is used for updating the neurons' weights subsequently by a technique called *Backward propagation*. The amount the weights are updated is referred as *learning rate*.

After all data is fed into the network, the error of the output layer (the difference between real result and predicted result) is "propagated back" from the output over the hidden layers to the input layer. The goal of this process is to adjust the weights in the way to minimize the error to give accurate predictions. The more training examples of the data are seen, the better this statistical estimator will describe the underlying data, and the better the neurons weight's will be adjusted to improve the model's performance on the data. The other way around, using less data for training leads to noisier updates of the neurons' weights and a perhaps less accurate estimator of the error gradient. However, these noisier updates can lead to a lower training time and also a more robust model.

Another type of neural networks are "Recurrent neural networks (RNN)", which are mainly used for data with a temporal component, or tasks where the previous state has an influence on the next state, for example stock market predictions. RNNs are based on the idea of storing the previous state in so called "state matrices", which are used for predicting the next output.

The experiments of this work are all based on image classification. Therefore, the type of neural network used is the convolutional neural network.

2.3.1 Convolutional neural networks

A further type of neural network is the "Convolutional Neural Network (CNN)". It follows the same principles as the MLP, but it also includes so called convolutional layers. CNNs are often used in computer vision, processing images or videos. Images are represented pixel-wise by grids of numbers, usually using one single grid for grey-scale images and three grids for colored images. Each numeric value in a grid represents the intensity of the corresponding pixel. In the convolutional layers, a relatively small convolution matrix sized n*n neurons (typically n=2) is "moved" over the grid, filtering the image for noteworthy patterns (e.g. edges or shapes). It is usually followed by a pooling layer. This layer reduces the dimensionality of the data by combining outputs of several neurons into a single output by applying a filter. A very commonly used filtering technique is called "Max Pooling". Here, an n*n cluster is reduced to a singular value representing the maximum of all processed neurons. Alternative approaches are either using an average or median value or applying a majority voting. The goal of this filtering is to only keep the most important information, and to prevent overfitting and decreases the model's size, allowing more deeper and complex models. The biological pendant is called "lateral inhibition".

A graphical representation of the functionality of a (two dimensional) convolutional layer is depicted in Figure 2.4.

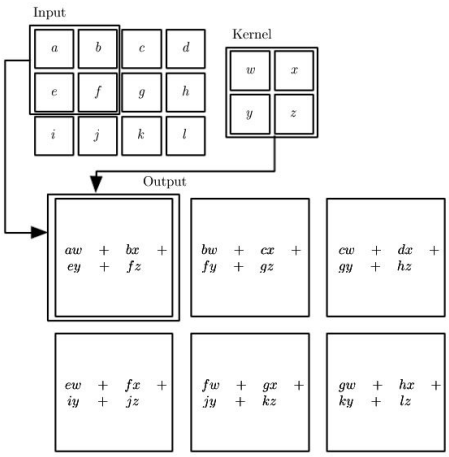


Figure 2.4: A figure visually describing a convolutional layer by Goodfellow et. al. [GBC16] (only processing values that are entirely inside the kernel)

2.3.2 Optimizing neural networks

There are several ways to optimize neural networks, e.g. by tuning important hyper-parameters such as batch size and learning rate or altering the network's structure itself.

Batch size

The size of a "batch" fed into the algorithm at one point of time, before the estimator for the error gradient is calculated, is called "batch size". It can of course be varied and acts as an important hyper-parameter of the model.

The selection of the value of the batch size are usually driven by the following factors summed up by [GBC16]:

- Larger batch sizes often result in a higher accurate gradient-estimator with less than linear returns.
- Observations showed that modern multi-core architectures are often underutilized when using very small batches, resulting in a minimum batch size where reduction does not reduce process time.
- On the other end of the spectrum, when all data of a batch is processed in parallel (which is done in most cases), the amount of memory needed scales with the size of the batch, therefore acts as a limitation factor.
- It is common that some batch sizes, especially those to the "power of two" run faster on some GPUs . Typically, they range from 32 to 256.
- Small batch sizes are often noisy, leading to a regularizing effect and lower generalization error. The more overfitting occurs, the bigger the generalization error.
- The lowest generalization error is often achieved when using a batch size of 1. But at a such low value it is also recommended to lower the learning rate, because the variance of the estimated gradient is likely very high, which would lead to an unstable model. Also, it takes long to see the entire dataset. Therefore, this setting often results in a very high training time.

Learning rate

Learning rate, also called step size, is a hyperparameter (usually between 0 and 1) that defines the size of the model's weight updates. The higher this value is, the faster the network adapts to the seen data - in other words, the less training epochs are required. A smaller learning rate adapts the network weights slower, resulting in a higher amount of training epochs needed. However, a learning rate that is too high can lead to an under-performing model that converges too quick, while a low learning rate can lead to a stuck training process. Finding an optimal learning rate depends on the concrete architecture of the neural network but is a high factor for success.

Neural network architecture

Each year from 2010 to 2017, research teams competed in the ImageNet Large Scale Visual Recognition Challenge [RDS+15], a challenge evaluating the reseachers' algorithms on a big dataset of labelled images, trying to achieve the highest accuracy.

Out of the 2012's challenge, one of the most popular deep convolutional neural network algorithms, called AlexNet [KSH12] came out as winner. This algorithm is of high importance, because the team around the author Alex Krizhevsky, and his PhD advisor and later Turing award winner Geoffrey Hinton used some, by then, radical new approaches in the field of neural network design. In the following, the key parts of the algorithm are described.

AlexNet's architecture consists of eight layers in total: five convolutional layers followed by three fully-connected layers. The last fully connected layer is fed to a 1000-way softmax, an activation function which turns logits into probabilities, and produces a distribution over 1000 classes. In total, the network possesses over 60 million parameters and 650 million neurons. It is depicted in Figure 2.5.

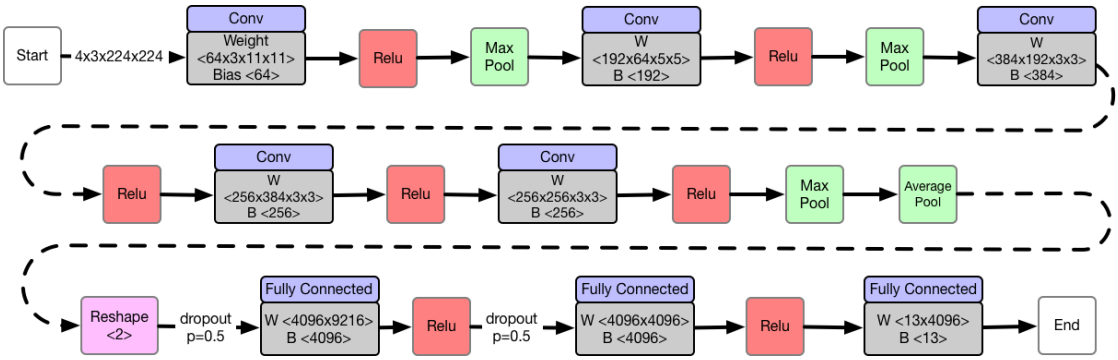


Figure 2.5: The deep neural network "AlexNet"

They built their architecture to be capable of being trained in parallel by multiple GPUs, allowing a bigger model to be trained in less time.

The algoritm uses a non linear activation function, known as Rectified Linear Unit (ReLU). The team around AlexNet claims that it runs much faster than the more popular TanH activation function, which was standard back then.

Also, overlapping Pooling was introduced. In "normal" pooling, once these n*n batches were summarized, pixels of these batches were never touched again. Not so in overlapping pooling, where they are read in an overlapping way. As a result, pixels can be read more than once as they are read in more than one pooling step.

Next they addressed the problem of overfitting by applying data augmentation and dropout methods. Data augmentation means enlarging the dataset by for example scaling the images' size or by adding reflections, transformations or performing Principle

Component Analysis on the RGB pixels with a goal of altering the intensities. Using dropout means setting the output to 0 of each hidden neuron at a particular probability, in this case 50 percent. The goal of this is to force each neuron to have more robust features.

This algorithm won the 2012 challenge with a top-5 error rate of 15.3%, while the second best algorithm only reached an error rate of 26.2%, outperforming it my more then 10%.

Since 2012, numerous and often more complex deep convolutional neural network algorithms were invented, which increased the accuracy on the ImageNet dataset steadily and soon outperformed AlexNet remarkably.

With the rise of mobile devices with limited computational power, storage and power supply, some researcher start to shift their focus on reducing a model's size for a given accuracy level. For example, in 2016 an architecture called "Squeezenet" [IMA+16] was developed. It achieves the same accuracy on ImageNet dataset as the previously mentioned "AlexNet", but contains 50 times fewer parameters. They were able to compress the model to a size less than 0.5 megabytes, which is 510 times smaller than the size of "Alexnet". Such smaller neural network architectures are especially useful for federated learning because of the clients' limited resources and to limit the amount of exchanged data over the network.

2.4 Federated Learning

Federated learning is the process of decentralizing the training model generation. Rather than training all data on a big endpoint it is outsourced to many smaller endpoints.

It should also be capable of dealing with the following properties [KMRR16]:

- Non-independent and non-identically distributed data (a users' data is not necessarily a representation of the population's distribution)
- Unbalanced data (some users might produce more data than others)
- Massively distributed data and users
- Limited communication of endpoints

In the following, two major federated learning techniques on how to combine local models are described.

2.4.1 Sequential learning

In sequential learning, sometimes called "Cyclic Institutional Incremental Learning" [SRE+19], the model is passed from one node to the other. The subsequent node continues training from the state the previous node has ended its training. This is performed in a

number of cycles. A variation is to store the model in an intermediate node, where each client fetches the model from before training and sends it there afterwards. A graphical representation of the latter method is depicted in Figure 2.6.

Note: for sequential learning, we define the "cycle" for one iteration of the above mentioned learning cycle. After one client finishes its training on its local data, the model is handed over to next next client, until all clients have participated once, respectively all data is trained once.

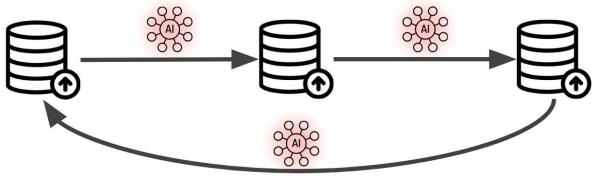


Figure 2.6: An illustration of sequential federated machine learning

2.4.2 Federated averaging

A second method for the clients' models is the aggregation through Federated Averaging. In this approach, the value for each model parameter is averaged over all clients in each round and stored in a common global model, which is redistributed subsequently.

Federated averaging can by categorized, according to Yang et. al. [YLCT19] into different types, based on the distribution characteristics of the data.

Horizontal federated learning describes data being distributed over several clients, where each of the datasets has the same features, and data only differs in its samples, also referred as observations. The goal of this horizontal federated learning approach is generating a common data model based on the different observations from all clients. In the medical domain, an example for horizontal federated learning is the same medical test (e.g. blood analysis) taken by many different patients in different medical centers. Each is taken at a different medical institute but by the same patient.

Vertical federated learning, also called feature-based federated learning, describes that the distributed data across the network shares the same observations, but each client observes different features of them. The goal of vertical federated learning is aggregating these different features originating from different clients and creating a model based on the data from all parties. In the medical domain, an example for vertical federated learning are different medical tests (e.g. computed tomography, blood analysis or allergy tests). Each is taken at a different medical center but by the same patient.

Federated Transfer Learning is a combination of the two above mentioned approaches, assuming that data not only differs in its samples but also differing in features at the same time.

In this work we will focus on horizontal federated learning and by using the phrase "federated averaging" we will always refer to horizontal federated learning through aggregation, if not specified otherwise.

Most of the current (horizontal) federated averaging algorithms consist of the following steps [KMY+16]:

1. Every training round, each client trains a local model based on their local data.
2. The clients send their locally trained models to a coordination server server
3. The coordination server combines the received models to create a new, improved global model, for example through averaging the models parameters
4. The new, updated model, is sent back to clients, and the process starts over, until the number of maximum rounds is reached or the loss function converges

This principle is also depicted in Figure 2.7.

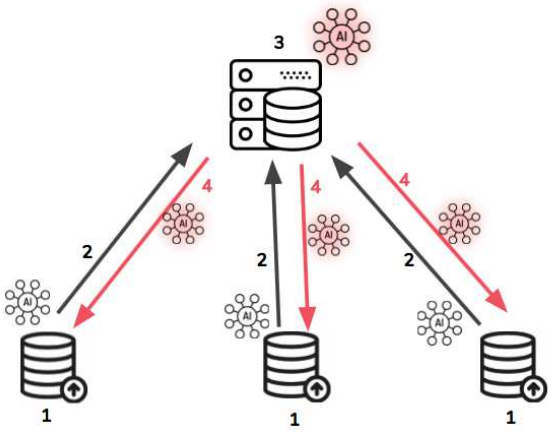


Figure 2.7: A typical network for (horizontal) federated machine learning

Also note that this algorithm is independent of specific machine learning algorithms (e.g. regression or neural networks). From a security point of view, it can be assumed [PAH+17] that the coordination server is *honest-but-curious*, meaning to be curious in extracting the data of individuals and being honest in operations, and the clients are honest in general - an assumption, which we will exploit in the next section of this work.

2.5 Overview of attacks on machine learning

In the perspective of adversarial machine learning, possible attacks are divided into the dimensions of the CIA triangle, as introduced in Section 1.2. The first of the three

dimensions, confidentiality, controls access to information. In the domain of machine learning, it restricts the access to a model only to those who are permitted. Integrity assures that a ML model is accurate, trustworthy and not altered. Availability guaranties a reliable and constant access to a model. A graphical representation is displayed in Figure 2.8.

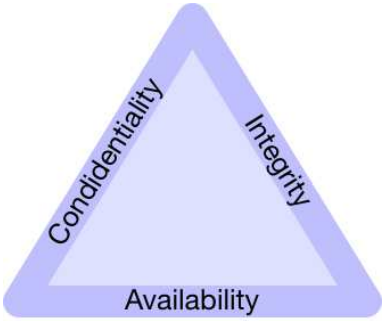


Figure 2.8: The CIA triangle

All three dimensions are examined for possible attacks and split into training and deployment phase.

2.5.1 Confidentiality

Training phase

In the training phase there is no known attack in terms of confidentiality.

Deployment phase

Model stealing
Machine learning models can be the target of stealing attacks, e.g. in Stealing Machine Learning Models via Prediction APIs [TZJ+16]. Stealing these models can be a valuable goal, especially when they are used in security applications like face detection for granting access to buildings or use sensitive training data or have a high commercial value.
Model inversion
Using the model inversion attack [FJR15] one can extract data from the training dataset. E.g. in neural network processing images, one can obtain the original image of the training set by simply knowing the name of the person.
Membership inference attack
The intend of a membership inference attack [SSSS17] is to gain the knowledge whether or not a particular example was in the training dataset.

2.5.2 Integrity

Training phase

Poisoning
Poisoning attacks try to change the behaviour of the model during the training process as the attacker would like to. A more detailed description is given in Section 2.6.

Deployment phase

Evasion
These attacks try to create adversarial inputs by adding noise to some data, which is, in best case, very small and not noticeable for the human eye. This altered data is then fed to the model during deployment stage, with the goal to achieve a misclassification. For more details see also Section 2.7.

2.5.3 Availability

Training phase

Poisoning
Poisoning attacks can also be used for weakening a models availability. An attacker aims to inject a large amount of "junk" data into the model that it becomes basically useless.

Deployment phase

Increasing false positives
The goal of an attacker might be misclassification, again realized by evasion - e.g. flooding the model with wrong predictions.

In this work we put our focus on integrity based attacks which are explained in more detail in the following sections.

2.5.4 Categorization by model access and goal

Adversarial attacks can also be categorized in two different ways, namely the amount of access to the model and the actual goal of the attack. While these definitions sometimes vary throughout different papers, we are referring the following definitions to [AT19]:

Categorization by model access
white box attack: an attacker has access to the model and also has all knowledge on the model's architecture, parameters and input feature representations.

black box attack: an attacker only has access to the inputs and outputs, but does not have knowledge on the model's internal structure, parameters or feature representations.

grey box attack: an attacker has only access to some information, e.g. only knows the overall structure of the network or the activations of the last layer.

Categorization by goal
confidence reduction: do not change the output class of a prediction, but alter the uncertainty.

source (untargeted) misclassification: changing the output class of an input to a different, random class.

targeted misclassification: changing the output class of a single input observation to a specific target class. This is the goal of the usual evasion attacks, trying to trick the model for one specific input by adding noise.

source/target misclassification: changing the output class of a certain input class source to a specific target class. An example for this attack is changing all observations from (input) class "2" to an (output) class "5".

2.6 Poisoning attacks

Poisoning attacks have been around since around than 10 years, e.g. used in [BNL12]. Here, adversaries try to manipulate the data that is used to train the model with the goal to alter the model's behaviour in their own desire, e.g. to lower confidence or force misclassification.

Poisoning attacks recently gained attraction when the so called "backdoor attack" was introduced in a paper by Gu et al. [GDG17]. These attacks aim to cause *targeted misclassifications* by adding specific patterns to the data that is used in the training process. While maintaining a high performance on the benign (original) test samples, all samples assigned with the backdoor trigger shall be misclassified to a predefined class.

In contrast to lowering confidence done in prior attacks, in a BadNet a model's owner might need more time for detecting backdoors in a model fully functioning on (benign) input data, than she needs for detecting that model is not working at all.

Beside adding the backdoor via altering the training data for the model, a second strategy is to make users use a previously learned poisoned model using transfer learning. An advantage the latter method has is that transfer learning is currently the most used method for recognising images, according to Gu et al. [GDG17]. There exist many pretrained models for most popular neural network architectures. An attack for this is also described in above mentioned paper, where the authors train a neural network on an US American traffic dataset. The data the model is trained with was equipped with some backdoor patterns. The authors then retrain the same model on the Swedish traffic sign dataset, and the backdoor still manages to survive..

2.6.1 Backdoor insertions strategies into the global model

In the following, we describe the two most used attack strategies for entering backdoors into a federated averaging setting in our processed literature (see Sections 2.6.2 and 2.6.3)

Basic Attack strategy
The basic strategy follows a very naive approach and can be used for sequential learning as well as federated averaging.

In sequential learning, the global model is gathered in the beginning. If a malicious client gets the model, he trains it with a fraction of poisoned data mixed with benign data. Notably, the training process uses the same hyper-parameters as a benign client would use. Finally, the model is handed over to the next client.

In an federated averaging environment, all benign and malicious clients train their models at one step of time. Note that the fraudulent client again trains his model using a fraction of poisoned data mixed with benign data. After the training is finished, all local models are simply handed back to the aggregator and their parameter updates are averaged.

As a result, using this attack strategy, a malicious client participates exactly the same way as a benign client does on the global model.

Model replacement attack strategy
Model replacement strategy is a more advanced approach, based on the methodology of [BVH+18]. Their experiments using the basic attack strategy resulted in a very low influence in terms of effectiveness of the adversary on the global model, and it takes a lot of time to do little progress. The reason is that most of the malicious client's updates is lost during the averaging process.

This strategy tries to manipulate the "size" of the adversary's parameter updates with the goal of surviving the averaging process. In fact, the adversary even tries to fully replace the joint (global) model entirely by the local malicious model. Note that this strategy only works in the case of joining the models using federated averaging.

In the following, this procedure is covered in detail. Formally, the aggregation of all local models into the global model using federated averaging can be expressed as defined in Equation 2.1.

$$G^{t+1} = \frac{1}{n}\sum_{i=1}^{n}(L_i^{t+1}) + \frac{1}{m}\sum_{i=1}^{m}(L_i^{t+1}) \tag{2.1}$$

G^{t+1} describes the global model in the next epoch (t+1), and L_i^{t+1} a local model i in the next epoch. n equals the number of benign clients, and m the number of malicious clients.

By using basic algebraic transformations, we can rewrite equation 2.1 as 2.2.

$$G^{t+1} = G^t + \frac{1}{n}\sum_{i=1}^{n}(L_i^{t+1} - G^t) + \frac{1}{m}\sum_{i=1}^{m}(L_i^{t+1} - G^t) \tag{2.2}$$

Here, the global model is subtracted from all local models and summed up, and afterwards the previous global model is added again.

Using this representation we can assume, that the sum of deviations of all benign local models with respect to the previous global model will converge to 0 (equation 2.3) - especially if the training process is advanced, after many epochs. The reason is that the global model is likely to converge by then, and all the local models are based on non-i.i.d data. This is formulated in equation 2.4.

$$\frac{1}{n}\sum_{i=1}^{n}(L_i^{t+1} - G^t) \to 0 \tag{2.3}$$

Therefore:

$$G^{t+1} = G^t + \frac{1}{m}\sum_{i=1}^{m}(L_i^{t+1} - G^t) \tag{2.4}$$

As seen in equation 2.5, our goal is to replace the new global model G^{t+1} by our local poisonous model, we replace this term by P, representing the latter. Also, for simplicity reasons we assume that the attacker who wants to replace the model consists of only one client. This leads to:

$$P = G^t + \frac{1}{n} \cdot (L^{t+1} - G^t) \tag{2.5}$$

By solving the resulting equation for the poisonous updates represented by L^{t+1} we get equation 2.6.

$$L^{t+1} = n \cdot (P - G^t) + G^t \tag{2.6}$$

Intuitively, this means that we have to simply scale up the poisonous model's differences by the number of clients that participate in the federated network. This scaled up version is than handed back to the aggregation process to fully replace the global model by surviving the average.

Both strategies are state of the are ways to introduce backdoors into federated learning through federated averaging. In the following, some successful backdoor attacks used in literature are described.

2.6.2 Attacks in a centralized training environment

Two successful attacks from the past are listed in the following.

Observations by Gu et al. [GDG17]

Gu et al. test the principle of backdoor attacks on the "U.S. traffic sign dataset", a dataset containing 8,612 images at three classes. After training a convolutional neural network (CNN) with the originally labelled data, they duplicate a subset by randomly selecting 10% (which turned out to be enough) of the original dataset and modify it with backdoor patterns. They use three different shaped backdoors (a yellow square, a flower and a bomb) with dimensions "similar to Post-it notes on a real-life traffic sign". See also Figure 6.3 in Chapter 6.

After adding backdoors to the training data, two different labelling strategies for backdoored images follow. The so called "Single target attack" changed the label of a backdoored stop sign to a speed-limit sign, while the "Random target attack" changed the label of an arbitrary traffic sign with a backdoor to a randomly selected incorrect label, with a goal to decrease classification accuracy when backdoors are present. Finally, the CNN is retrained with all backdoored images.

Their observations concluded that more than 90% of the backdoored images in case of the "Single target attack", and around 98% in case of the "Random target attack" were successfully labelled incorrect, while the accuracy of correctly labelled non-backdoored images was still nearly as high as before.

Gu et al. also use backdoors on the MNIST [LC10] dataset. This set of data contains 60.000 handwritten digits as 28x28 greyscale images. Again they start their experiments with a pretrained deep neural network and select p% of the training dataset, where they add the backdoor in form of a specific pixel pattern. Their results show that even the smallest tested value of p=10% leads to significant results.

The goal is to misclassify all numbers with added backdoors to represent a specific different number (this scenario is tested for all ten digits as target classes one-by-one). On benign images, the error rate (error rate = 1 - accuracy) is at most 0.17% higher than, and in some other cases even lower than the error rate of a non-backdoored CNN. The error rate for backdoored inputs is at most 0.09%, meaning that more than 99.9% of the backdoor images are successfully misinterpreted.

Observations by Wang et. al. [WYS+19]

Wang et. al. show successful backdoor attacks in a variety of image classification benchmark datasets. They add white squared backdoors into the bottom right corners of the images. Also, they make sure that the patterns (shape and color combination) do not occur naturally in untouched images. Furthermore, they limit the size of these patterns to 1% of the entire image. On each tested dataset, they achieve a 97% success rate of the attack, while losing at most 2.62% accuracy on the benign data across all datasets. All their experiments are hold in a central machine learning setting, at a malicious data ratio between 10% to 20%, depending on the dataset.

2.6.3 Attacks in a federated averaging environment

In the following, some work on attacks in an environment with federated averaging are listed.

Observations by Bagdasaryan et. al. [BVH+18]

Bagdasaryan et. al. test backdoors in a federated averaging environment based on two datasets, the "CIFAR-10" dataset [KNH12] for an image classification task, and the "Reddit" dataset [MMR+17] for a word prediction task. The federated network consists of 10 clients. They experiment with different model parameters, attack models and try three different kinds of backdoors. Each client locally trains its model for 100 epochs, at a batch size of 64. They use the same two attack models (see Section 2.6.1) as we later use in our experiments.

Each of their malicious clients use a batch size of 64, and each of these batch sizes contains only 20 backdoored images and 44 benign images. While this relation is kept constant, we vary this ratio later in our experiments

Their experiments are split into two different timing strategies for their attacks: the "single shot attack" and the "repeated poisoning attack". In the "single shot attack", one malicious client is only used for the training in one point of time (and ignored in the rest of the epochs), while on the "repeated poisoned attack" they are selected each round.

To be able to compare results, we only consider results from the "repeated poisoning attack" in the image classification task. They test a variety of different percentages on backdoored clients, namely 1%, 2%, 5%, 10%, 20%, 50% and 100%.

To achieve a backdoor accuracy of around 50% with two of their three tested backdoor patterns, the backdoored clients need to make 20% of the total population in case of the basic attack model. For an accuracy of over 90%, backdoored clients even have to make up more than 50% of the total clients.

For the model replacement attack, only around 1% malicious clients are needed for an accuracy of around 50%, and around 5% to reach the 90% accuracy threshold.

Observations by Sun et. al. [SKSM19]

The authors experiment with backdoor attacks in a federated averaging environment. In contrast to earlier work, the authors also consider that benign clients contain data with the backdoor pattern, however correctly labelled (as they would be classified without the backdoor pattern). Parameters they are investigating are the fraction of benign to malicious clients and different backdoor sizes. Experiments are hold on a handwriting dataset. They only consider the model replacement attack, as described in Section 2.6.1. Defensive strategies tested are a variant of "Differential Privacy" (see Section 2.8.2), and the rejection of updates which norm is above a threshold (to counter "boosted" updates coming from the model replacement attack).

They result that for non-defended networks the performance of the backdoor attack mostly depends on the fraction of malicious clients, as well as the size of the backdoor.

Previously mentioned defensive strategies, however, are enough to mitigate backdoor attacks, without significantly decreasing the performance on the benign task.

Observations by Nguyen et. al. [NRMS20]

Nguyen et. al. also evaluate backdoor attacks in a federated averaging, on three different real-world IoT datasets. Investigating different parameters such as a varying percentage of malicious clients, the authors focus on evaluating an attackers success in a network with active defensive measurements. Precisely, they test in a network with "Generalized clustering approaches" (see Section 2.6.4) and using "Differential Privacy" (see Section 2.8.2).

They conclude that these current mitigation techniques are ineffective as they are able to successfully introduce poisoned data without being detected. Also, they propose three different new areas of potentially effective defensive strategies that are left to be investigated by future work.

2.6.4 Defense strategies against poisoning attacks

Defense strategies against poisoning attacks in central machine learning settings exist in large numbers. Most of them are based on either a statistical analysis of the poisoned training data or analyzing the activation of certain neurons on a dataset.

In the following, some approaches of literature are listed.

Steinhardt et al. [SKL17] propose a defensive mechanism that is based on outlier detection and removal. The authors develop a new algorithm for this task, in which they construct a (theoretical) upper boundary for the loss of model updates, rejecting potentially malicious updates with the loss increasing this threshold.

Liu et. al. [LXS17] propose three different mitigation techniques that filter the inputs (prior to classification) by input anomaly detection, re-training, or input preprocessing.

Later, Liu et. al. [LDG18] propose a solution that is based on pruning. Experiments showed that backdoored inputs trigger different neurons which would be dormant in the case of benign data. They showed that by removing those neurons that were dormant for clean inputs, they are able to reliably reject malicious inputs.

All of the above mentioned approaches have been used to defend poisoning attacks. But all of them are based on the following assumptions. They require the knowledge on a set of data that can definitely be trusted and having insight on the training data.

Detection based on data provenance:
A recent procedure, targeting poisoning attacks by using data provenance is proposed by Baracaldo et. al. [BCL+18]. In contrast to above mentioned method, this algorithm

does not need a trusted set of data. The authors' method works in environments with data being partially trusted, as well as in environments without having trusted data at all. In the following we describe the algorithm used in the partially trusted environment. Starting point is the availability of a trusted training set as well as an untrusted training set, a trusted "provenance dataset" containing metadata on the origin of the untrusted data and a "provenance feature" to cluster data points by a certain property (e.g. the factory the data originated from).

In the first step, each untrusted datapoint is linked with its corresponding record in the provenance dataset. Next, the untrusted dataset is segmented into n smaller fragments by the selected provenance feature. Then, *n-1* fragments are trained using a machine learning algorithm ("filtered model"), excluding one fragment that is targeted for an analysis to determine whether it is poisonous or not. By comparing this "filtered model" with a model trained on all available data ("unfiltered model") on the trusted testset, it is determined if the corresponding segment is poisonous or not - precisely, if the unfiltered model is performing worse (above a threshold) than the filtered model. This procedure is visualized in Figure 2.9.

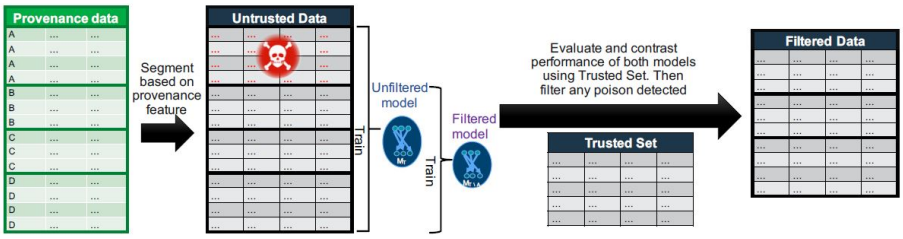


Figure 2.9: A graphical representation of the procedure by Baracaldo et. al. [BCL+18] who also created this image

Above mentioned methods focus on mitigating classical poisoning attacks (with the goal of lowering the model's confidence or classification rate). Backdoor attacks, however, target to not hurt the performance on benign data but only on data altered with this pattern. The latter are not explicitly considered in all approaches listed above.

Detection based on activation analysis:
Activation Clustering [CCB+19], on the other hand, is explicitly capable of dealing with backdoor attacks in neural networks and does not require knowledge on trusted data. It is based on the idea that benign data only triggers features associated with its target class, while a malicious input triggers both features of their real class and their backdoored class. In detail, the network is first trained with (untrusted) data, which might also contain backdoors. Afterwards, the network is queried by this data again, but this time the activations of the last hidden layer are stored. Then, they are separated by the target classes, and after applying some dimension reduction function they are clustered (e.g. using k-means clustering at k=2). By doing so the data is separated into two clusters and all poisoned data points should result in one cluster. This approach however requires that (natural) outliers of the benign data set do not have a strong impact on the model.

A visualization of clustering "stop traffic signs" labeled as "speed limit signs" (orange squares on the graph) and correctly labeled "speed limit signs" (blue squares) on a traffic sign dataset is depicted in Figure 2.10.

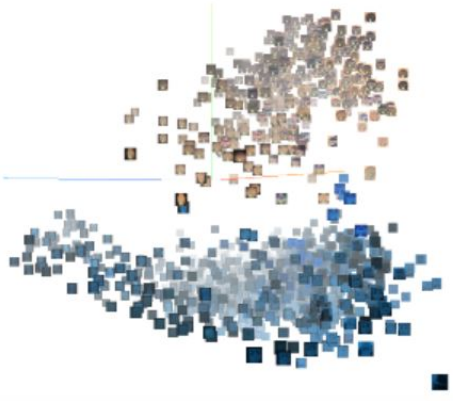


Figure 2.10: A graphical representation by [CCB+19] visualizing two resulting clusters of correctly labeled "speed limit traffic signs" and backdoored "stop signs" labeled as "speed limit traffic signs"

As the only algorithm explicitly dealing with backdoor attacks results that activation clustering is the most promising defensive mechanism against backdoor attacks in a centralized setting. However, when it comes to backdoors in federated learning, this defensive algorithm (as well as all of the above mentioned strategies) violate basic assumptions of federated learning. By design, in this special form of machine learning, the users' never publish their data. Also, a fundamental assumption is that the data of the clients is not independent and not identically distributed (non-i.i.d). Additionally, in the case of secure aggregation (see Section 2.8), which is de-facto standard in a federated learning network, clients do not even publish their local models.

Furthermore, according to [BVH+18], it is even provably impossible to detect anomalies in the models sent by the clients, unless there is a way for the secure aggregation protocol to check. Even if there was a way to implement such a check into the protocol, it somehow needed to filter only the backdoored models and not the benign model, while keeping in mind that the data being trained is non-i.i.d.

Summarizing, generally feasible mechanisms against backdoor attacks in federated learning that work in all cases without any prior assumptions are, to our knowledge, not available.

In the following section a different type of integrity based attack is described - so called evasion attacks. While this thesis focuses on backdoor attacks, an attacker can use a special type of evasion attack called "Universal Perturbation attack" to generate backdoor like patterns that are more "stealthy".

2.7 Evasion attacks

An evasion attack happens when a network is fed with an adversarial input, and was first described by Szegedy et. al. [SZS+14]. These inputs are equipped with some small, but deliberately added perturbations, with the goal to trick the model and achieve misclassification of the input at high confidence. Goodfellow et. al. [GSS15] state that this behaviour is a result of the model's linearity in high-dimensional spaces. Many different algorithms for generating these perturbations were invented, but also defensive strategies.

Note that in contrast to backdoor attacks, the attacker in general cannot change neither the training data nor the target model itself.

2.7.1 Overview over evasion attacks

Many different adversarial attacks exist and can be classified can be categorized by knowledge of the machine learning model as well as the goal of an attack (see Section 2.5.4). Each attacks has its own strengths and weaknesses regarding computation speed and amount of perturbations. Nicolae et. al. [NST+18], designers of the Adversarial Robustness 360 Toolbox, give an overview on a list adversarial attacks in their paper. They list over 20 different evasion attacks that differ in computation time, perceptibility or success rate.

2.7.2 Fast Gradient Sign Method

In the following, the first known evasion attack called Fast Gradient Sign Method by Goodfellow et. al [GSS15] is described in detail. Although the perturbing effect is much bigger than in newer approaches, many of them are at least partially based on this approach.

This attack can be classified as a white box and untargeted attack and only consists of single step attack (non-iterative).

In the beginning of this attack, the adversary sends an input image (which will be perturbed in the end) through the neural network to determine its correct output class. Then, one randomly selects a different output neuron that corresponds to a different class.

Now, gradient descent is applied to the input pixels of the image with respect to the classification loss of the randomly chosen class neuron, resulting in a gradient matrix. But instead of adjusting network weights in order to optimize ones classifier (which is normally done), the input pixels are adjusted to fool the network to make a wrong prediction. This is achieved by adding or subtracting a tiny (predefined) change to each of the pixel. Whether this error is added or subtracted depends to the sign operation of the gradient matrix. Note that the magnitude of gradient itself does not matter, only the direction (+ or -).

Finally, a small number (much smaller than 1, usually around 0.01) is multiplied to this matrix keep the resulting image as "close" as possible to the original image for the human eye. The resulting matrix is added to the input image, which is then finally perturbed.

A visual representation of this attack is presented in Figure 2.11.

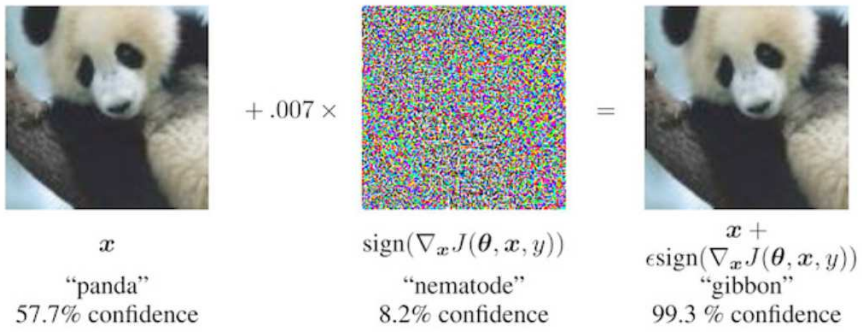


Figure 2.11: A graphic by Goodfellow et. al [GSS15], "transforming" a panda (left) into a gibbon (right) by adding a perturbation matrix (middle) multiplied by 0.007 to the panda image

2.7.3 Universal perturbation attack - a special evasion attack

Universal perturbation attacks represents a special case of evasion attacks. While usual evasion attacks aim to alter single data points during deployment phase, the attack proposed by [MDFFF16] aims to create an image agnostic perturbation with the goal of targeted misclassification. However, this pattern only requires a small training dataset to be generated and can also be applied to alter images which were not included in this dataset - as backdoor patterns.

The algorithm creating this image agnostic pattern works in an iterative way. For each data-point in the training set, the minimal permutation vector is calculated which, when added to the data-point, sends the data-point to the decision boundary of the classifier. This minimal permutation vector can be calculated by any existing evasion attack (e.g. Fast Gradient Sign Method).

After going through all the data points, these vectors are aggregated to represent the universal perturbation, tricking (almost) all data-points.

The whole procedure can be repeated several times to improve the perturbation's quality until it reaches a specified fooling rate or after a certain number of iterations.

Note that, as explained above, the Universal perturbation attack is a special attack which is placed "on top" of another perturbation attack, with the goal of finding a general purpose adversarial pattern for the neural network.

Zhong et. al. [ZLS+20] use a variation of this attack to create and inject a certain pattern before model training, at training a completely new model or as part of transfer

learning. The goal is to generate a targeted perturbation image that again enables targeted misclassification but is, in contrast to patterns used in backdoor attacks, for the human eye invisibly.

The next section deals with defending a neural network from evasion attacks.

2.7.4 Defense strategies against adversarial inputs

A profound analysis on defense-possibilities is done in [AT19]. Following them, defensive strategies can be grouped in 3 categories:

1. Modifying training samples
   Strategies from this category generate adversarial samples based on known attacks and insert this data correctly labelled into the training dataset, but label them correctly. The goal is to "robustify" the classifier.
2. Modifying model structure or training process
   The goal of approaches in this category is to train robust models, by modifying some parts such as number of layers, the format of the output layer or using robust loss functions. One could also add pre- or poststeps or training multiple times.
3. Combining with other models
   These strategies use multiple models (with different structures) to assist in the classification process

An overview on different defensive mechanisms belonging to one or more categories is given in [NST+18].

In the following we provide an overview on defensive strategies for confidentiality based attacks. Although they are not specifically part of our thesis, defensive strategies like "Secure Multi-Party Computation" are state of the art when it comes to model aggregation. As stated in Section 2.6.4, that's one reason why it poisonous updates are hard to detect.

2.8 Privacy threats in Federated Learning and strategies to defend

Melis and Song et. al [MSCS18] showed that clients exchanging model updates (with the coordination server) leak unintended information about their training data. There are several issues, especially concerning membership inference and property inference. Membership inference means that a malicious participant is able to determine if any given data point (an observation) was used to train the model. Property inference lets a malicious participant infer properties that only hold for a subset of the training data and are independent of features that characterize the classes of the joint, global model.

There are many different ideas on how to address these issues, including several encryption technologies to allow computation over encrypted data.

2.8.1 Secure Multi-Party Computation

Secure Multi-Party Computation enables one to perform computations on data of others while the data is still encrypted. Secure aggregation [BIK+17] is a technology in this field using a cryptographic protocol for aggregating parameter updates in a secure way. Here, the clients encrypt their parameter updates to ensure that the added information based on their input data is only learned by the coordination server doing aggregation. But it is still shown [MSCS18] that inference attacks can be successful even if an attacker only observes aggregated updates.

2.8.2 Differential Privacy

Differential Privacy [ACG+16] is the principle of adding noise to the data, scaling down model updates if they are higher than a set clipping boundary, or using some generalization methods to obscure some sensitive attributes of the data. This method can also be used to reduce the impact of poisoning attacks as poisoned updates are "averaged out". Usually, applying differential privacy to a network results in a trade-off between accuracy and privacy.

2.8.3 Homomorphic Encryption

Homomorphic encryption is a type of encryption allowing computations to be performed on encrypted operands, without the need of prior decryption, which again results in encrypted output. In a federated learning setting, the additive secret sharing technique described in [PSS07] seems very useful. At the beginning, one starts with two unencrypted tensors, which are encrypted by using encryption function - applied to each on its own. Then, one does the addition on the encrypted tensors and receives an encrypted result-tensor. Finally, one can decrypt the result-tensor, and achieve the same value as one would get by adding the two unencrypted tensors.

2.9 Existing frameworks and implementations

This section gives a short overview on different technologies in the field of machine learning, as well as attacks on machine learning models.

2.9.1 Federated Learning frameworks

From an implementation perspective, there are several frameworks for Federated Learning available, among them "TensorFlow Federated", "PySyft for PyTorch" and "Decentralized machine learning".

TensorFlow Federated [AAB+15] is based on the TensorFlow API, an open-source framework developed by Google, providing support for machine learning and other computation tasks like aggregated analytics on federated data.

PySyft for PyTorch [RTD+18] is an open-source Federated Learning library with its main purpose on building secure and scalable models. PySyft is based on PyTorch, a machine learning framework developed by Facebook.

Decentralized machine learning [Dec18] is a commercial, blockchain-based decentralized machine learning protocol.

PySyft and TensorFlow Federated are available under open source licences. PySyft is used in some projects, e.g. in [BVH+18]. Information about Decentralized ML is quite sparse, however they have recently announced that they are planning to release an app in the health domain showcasing their tools.

There are several other domain specific approaches, like [SGR+19], an application of Federated Learning for "Subcortical Brain Data", or [BK18], an approach for discovering user behaviour.

2.9.2 Adversarial Input Generation

There exist several frameworks for generating adversarial inputs.

Adversarial Robustness 360 Toolbox [NST+18] is an open source project by IBM, implementing several state of the art evasion attacks and detection algorithms. Furthermore, they also implemented backdoor attacks (following Biggio et. al. [BNL12] and Gu et. al. [GDG17]) and also defensive strategies against them (activation clustering proposed by Chen et. al. [CCB+19] and data provenance method by Baracaldo et. al [BCL+18]). It offers several robustness metrics, certifications and verifications. Its supports many machine learning libraries, among them PyTorch, TensorFlow, Keras and scikit-learn.

Advbox [Xl19] is, similar to the IBM Toolbox, a set of tools used for the generation, detection and protection of adversarial examples. It supports PaddlePaddle, PyTorch, Caffe2, MxNet, Keras, and TensorFlow.

CleverHans [GPM16] CleverHans is a Python library that tests different adversarial attacks on machine learning models and benchmarks them. It will support JAX, PyTorch, and Tensorflow 2.

CHAPTER 3

Methodology

3.1 Introduction

The goal of this thesis is to answer the research questions defined in Section 1.3, therefore gain insights on which parameters influence the effectiveness of federated machine learning and the success of backdoor attacks.

In order to address our discussed problem, we chose the methodological approach of design science [HRM+04]. The first step of this approach is the design of an artifact. In our case this refers to the setup of a testing environment, the selection of benchmark datasets, the creation of experiments and the implementation of neural networks used for classification. The second step of this approach, the evaluation of our artifact, is done against already existing artifacts. The latter are found in literature on the respective topics and are compared especially regarding effectiveness.

3.2 Datasets

Experiments are performed on the following datasets. They are selected because they come from well known domain machine learning domains and have both been already used for classification tasks. Moreover, the German Traffic Sign Recognition Benchmark has already been used in existing literature addressing backdoor attacks (e.g. by Wang et. al. [WYS+19]), which enables comparability to our work.

All created datasets, splits of the data, the models and result files are published at zenodo, and their corresponding DOIs are given in the later sections.

3.2.1 The German Traffic Sign Recognition Benchmark

The German Traffic Sign Recognition Benchmark is a single image, multi-class classification dataset on traffic signs. It consists of more than 40 classes and more than

50,000 observations of traffic signs. Image sizes vary between 15x15 to 250x250 pixels. The actual traffic signs are not necessarily squared and contain a border of around 10% around them. State of the art neural networks build by [BHN+18] are able to classify around 99% of the images correctly.

The classes of the this dataset are distributed very unevenly, as shown in the class distribution graph in Figure 3.1.

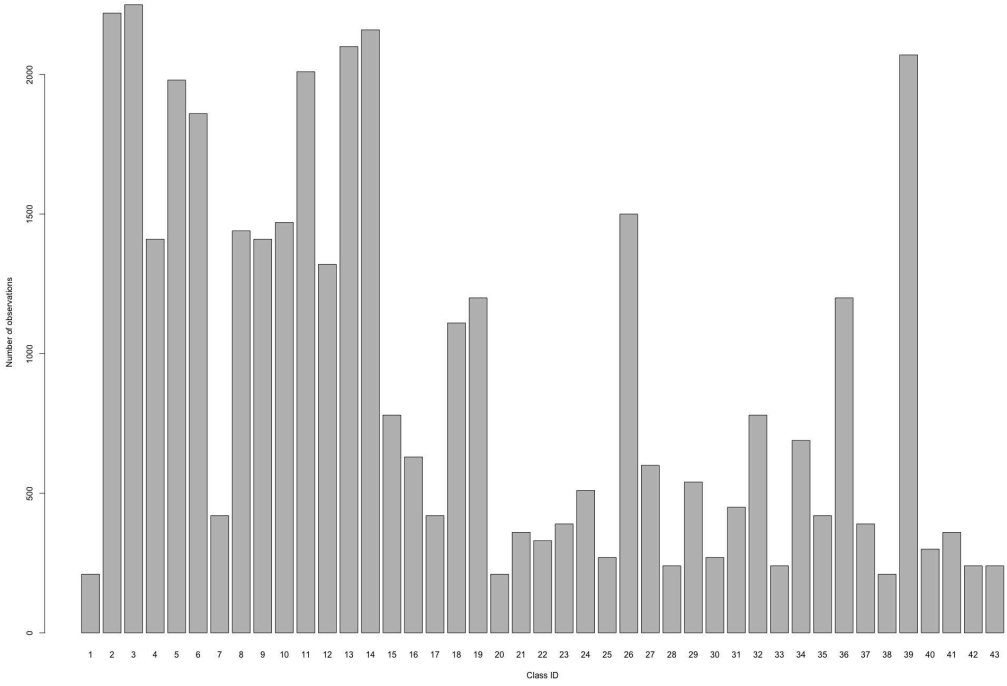


Figure 3.1: Histogram of the classes of the German Traffic Sign Recognition Benchmark dataset. Numbers map to the traffic signs displayed in Figure 3.2

To enable training in our PyTorch environment and keep comparability with state of the art work in [BHN+18], we also decided to resize all images to 32x32 (while applying distortion if needed). This size was chosen as a trade-off between the distribution of the given image sizes and the computational load. An overview of all classes is given in Figure 3.1.

As Wang et. al. [WYS+19] test white squared backdoors at a size of 1%, located on the bottom right corner, we use squared size patterns as well. We are also varying sizes (1% and 0.5% of the signs area) and colors of these backdoors, and place them in a squared area of around 30% in x and y direction around the center of the images, to ensure that the patterns are on the actual traffic sign.

Figure 3.2: All classes of the German Traffic Sign Recognition Benchmark dataset

The patterns are automatically implemented via a self-written Adobe Photoshop script. A sample outcome of the script on an image of the class "sharp right turn" using a 1% sized green squared backdoor is depicted in Figure 3.3.

(a) Image without backdoor (b) Image with backdoor

Figure 3.3: Results of backdoor insertion into the German Traffic Sign Recognition Benchmark dataset

3.2.2 The Yale Face Database (Facial Recognition dataset) [PNBK97]

The Yale Face Database is a collection of faces of 15 different individuals in greyscale images. Each observation of a person shows the person's face at a different facial expression. Possible configurations are "center-light", "happy", "left-light", "(no)glasses", "normal", "right-light", "sad", "sleepy", "surprised" and "wink". Each image has a size of 285x224 pixels. To enable comparability with the results by [KMT13] we also resized the images to 64x64.

All 15 individuals are displayed in Figure 3.4, are evenly distributed and have the same number of observations.

To increase the size of the training and test set, we apply the following augmentation techniques: a horizontal flip, and a change of brightness by -60%, -30%, +30% and +60% percent. Each of the brightness changes is done on the original image as well as on the horizontally flipped image. These augmentations are added after the dataset is split into train and testset.

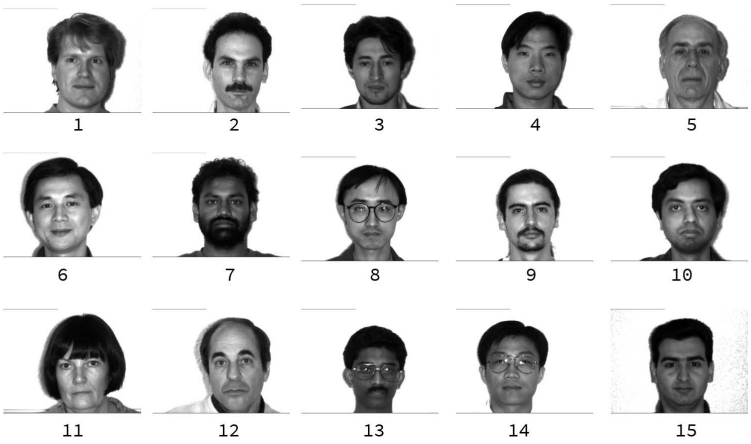


Figure 3.4: All individuals represented by the Yale face dataset

Different types of backdoors are tested on this dataset, both inserted by an iOS application called FaceApp [Fac19]. This is a commercial application for recognizing facial features by a machine learning algorithm. An example for an implemented backdoor is this specific kind of glasses shown in Figure 3.5.

(a) Image without backdoor (b) Image with backdoor (glasses)

Figure 3.5: Original image vs. image with backdoor in shape of glasses

3.3 Algorithms

Different Convolutional Neural Networks, with configurations depending on the datasets are used for the experiments, based on literature or other published results. A central criterion for selecting neural network architectures is that it should be a simple network that potentially works in a federated setting. Addressing backdoor attacks, even though we test relatively simple neural network structures, backdoor attacks have shown in the past (e.g. [BVH+18]) that they are also present in more complex architectures as well.

The German Traffic Sign Recognition Benchmark [SSSI12]
The classification of traffic signs is a very common task in machine learning. Findings from these tasks showed that convolutional neural networks outperformed other machine learning algorithms. Bengston et. al. [BHN+18] did research on several neural networks used for classification of traffic sign datasets, and we implemented the best performing network, shown in Figure 3.6. The network consists of two convolutional layers, followed by a max-pooling layer, and again two convolutional layers again followed by a max-pooling layer. Then, three times and in alternating manner, dropout regulation is followed by a fully connected layer.

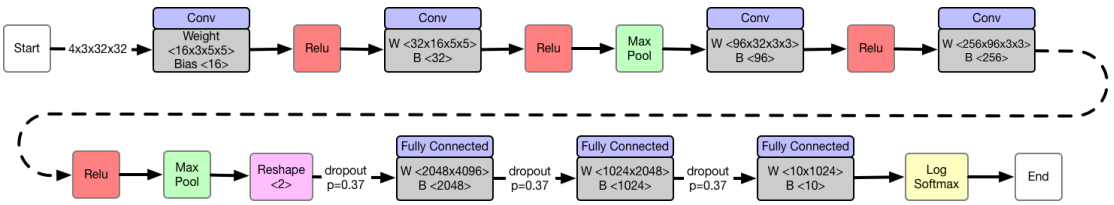


Figure 3.6: The deep neural network used for the traffic sign tasks

The Yale Face Database
On this facial recognition dataset we use the architecture by [KMT13]. This neural network has shown that it is viable for face recognition tasks in the past (e.g. on "AT&T Laboratories database of faces" [BBB13] or the "JAFFE face database" [LAKG98]), and also been used on our Yale dataset. The architecture is also depicted in Figure 3.7.

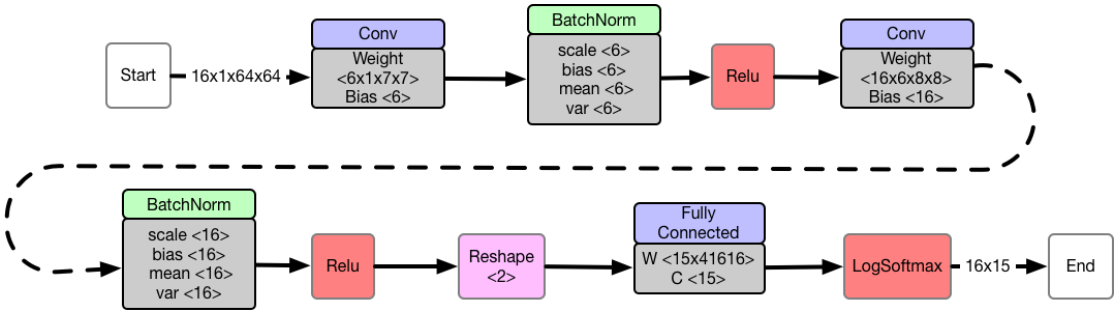


Figure 3.7: The deep neural network used for the face recognition tasks

3.4 Experimental workflow

The neural networks in our evaluation are built using the PyTorch platform, in combination with the PySyft framework that is used for the federation of the data, communication, model exchange and coordination.-

The experiments are performed using the following workflow:

1. Find a **good working configuration in literature**. Reproduce this configuration to perform as a starting position for further experiments.
2. Investigating the **impact of the parameters** listed below, by using above mentioned measurements. Each parameter shall be varied one by one, while keeping the rest at default value. Parameter values are selected to be in line with existing experiments in literature.

   The following experiments have a focus on the behaviour of federated learning in contrast to centralized machine learning:

   - number of **distributed sources**
     - one: 1 (equals central machine learning)
     - few: ≈ 5
     - many: ≈ 10
   - the **distribution of the data** over processing nodes
     - each unit has an equally distributed subset of the dataset
     - some classes are only known to specific units

   The following experiments have a focus on the behaviour of backdoor attacks:

   - the **appearance of the backdoor**
     - depends on datasets, eg. size, shape or color

- different **merge strategies**:
  - train all nodes at one point of time and afterwards do federated averaging
  - train them sequentially and update the global model after every training step of each individual unit
- the **percentage of backdoored nodes**
  - between 5 and 60%
- the **percentage of poisoned data in backdoored nodes**
  - 12.5%
  - 25%
  - 50%
  - 75%
  - 100%
- the **order of time** when backdoors are being trained. As we are using sequential learning, the order of presentation of the malicious nodes matters.
  - backdoored nodes are trained in the beginning of each learning cycle
  - backdoored nodes are trained in the end of each learning cycle
- the **attack model**. The attacker could simply train its model on backdoored inputs and send it back to the intermediate node, or she could use a more advanced strategy like a model replacement strategy (as described in 2.6.1) by scaling up the weights of the backdoored model to ensure that the backdoor survives the merging step.

3. **Evaluate the experiments** and compare to existing results from literature

3.5 Measuring results

The experiments are measured using effectiveness metrics. Effectiveness is measured through accuracy (overall or per class) on the test set. Accuracy is defined as the fraction of correct predictions in relation to the total number of predictions. Each of these measurements are done after a training epoch finishes.

Note that in the federated averaging case, we divide the training set into batches of numerous observations each and distribute them among all clients. Each (mini) training round, each client participates exactly one time to the updated model, by locally processing one batch of training data. One epoch is defined as the total time needed for the clients to process the whole data, which is split among all clients, exactly one time - therefore, usually consists of many (mini) training rounds. We use this definition of "epoch" in all experiments on federated averaging.

In Table 3.1 the different points of view of different actors are displayed. While a "normal" client aims for high performance on benign input data, input data with backdoor patterns

should result in a low accuracy. The attacker's goal is to increase the accuracy on backdoored samples, while still keeping a high performance on benign input data. In this work we put on the viewpoint on the attacker - in other words, we consider a high accuracy as "good" on benign as well as backdoored data.

Table 3.1: Targets of different actors in the network

	normal user	**attacker**
benign input data	high accuracy	high accuracy
backdoored input data	low accuracy	high accuracy

Note: The appearance of the backdoor patterns itself is not a primary concern. While the created backdoors are noticeable, they are chosen to be not suspicious as they naturally occur in the selected data. In the domain of traffic signs for example, a sticker on the sign does not raise a high amount of attention as many signs are equipped with stickers anyway. An example of this is shown in Figure 3.8. For the yale faces, our selected backdoors (a certain type of glasses and a full beard) are also realistically looking and not suspicious.

Figure 3.8: In image at a crossroads taken by Google Street View in the middle of Vienna [Goo20]

3.6 Working environment

The following environment is used used for the experiments:

Hardware
Intel Core i5-4690K, 4x3,50GHz
Nvidia GeForce GTX 970 4GB GDD
16GB RAM

Software
Windows 10 (64 Bit)
Python 3.7.3
pytorch 1.4.0
pysyft 0.2.3a2

All models are trained and tested in GPU mode using cuda 10.1. The minimum requirements for PySyft are listed in the "requirements.txt" file in the appendix (see 8.4).

3.7 Summary

In this section, we selected benchmark datasets for our experiments to investigate the research questions defined in Section 1.3. We chose a dataset in the domain of traffic sign classification and a dataset on facial recognition, two often processed problems in the domain of machine learning. For these datasets we examined relevant literature and extracted information such as neural networks and performance metrics on state of the art these datasets. According to the previously designed research questions we setup an experiment plan to determine which parameters are should be tested in the next step. In the following two chapters we implement concrete testing environments and perform tests according to our experiment plan. Finally, these results are to evaluated in Chapter 6.

CHAPTER 4

Case study on traffic sign classification

This chapter contains a case study using previously mentioned methodology on the German Traffic Sign Recognition Benchmark dataset. First, we review the state of the art this data is being processed, which is rebuilt in the following. Then, we test non-attack settings like the number of clients or data that is distributed in a non-iid way as a basis for understanding the parameter impact. Moreover, we perform backdoor attacks on this dataset with varying backdoor patterns, attack strategies, different numbers of attackers and points of time the attacker is selected in a sequential learning cycle. The goal of this is to provide an insight if (and how) backdoor attacks can be applied on this dataset.

4.1 State of the art

Bengston et. al. [BHN+18] do intensive tests on the "The German Traffic Sign Recognition Benchmark" dataset. Their best tested neural network architecture, depicted in Figure 3.6, is trained at for 75 epochs, with the "Stochastic Gradient Descent" optimizer at a learning rate of 0.005 and a categorical cross-entropy loss. At a batch size of 5, they received a test set accuracy of 99.35%, at a batch size of 10 an accuracy of 99.01% and at a batch size of 20 an accuracy of 98.67%.

For our experiments, we rebuild their neural network structure and use the same parameters to the best of our knowledge. One parameter we altered is the batch size - the number of training data fed into the algorithm at one point of time, before the estimator for the error gradient is calculated. As preliminary tests on the dataset have shown that the smaller the batch size is, the longer a training epoch takes to finish, but the better the overall model performance becomes.

Due to hardware limitations, we decided to use a much bigger batch size (n=512) for our experiments which decreases the training time drastically but lowers the model's performance to small extends. In Figure 4.1 we did testruns in a centralized settings using given batch sizes of 5 and 15, and our used increased batch size of 512. It turns out that smaller batch sizes perform better, however they take way longer to be completed. Each training epoch in the case of n=5 takes 180 minutes to train, at n=15 it takes around 61 minutes and at n=512 each epoch takes around 2.5 minutes. In comparison to 180 minutes (n=5) this equals in increase of speed by 7200% per epoch while losing 3% of accuracy.

Our dataset is uploaded to Zenodo and is achievable under the DOI *10.5281/zenodo.3716766*, as well as all our resulting models under the DOI *10.5281/zenodo.3723574*. The mapping of the models to the concrete experiments is described in the appendix in Section 8.1.

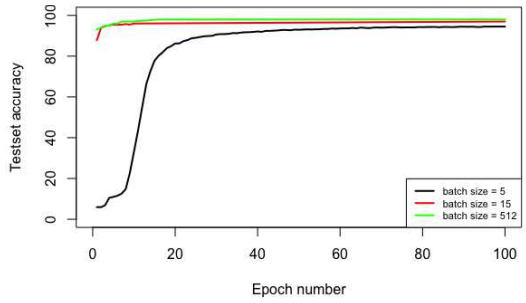


Figure 4.1: Rebuilding the state of the art neural network for the traffic sign classification using centralized learning on varying batch sizes

Table 4.1: Configuration used for experiments on the number of clients

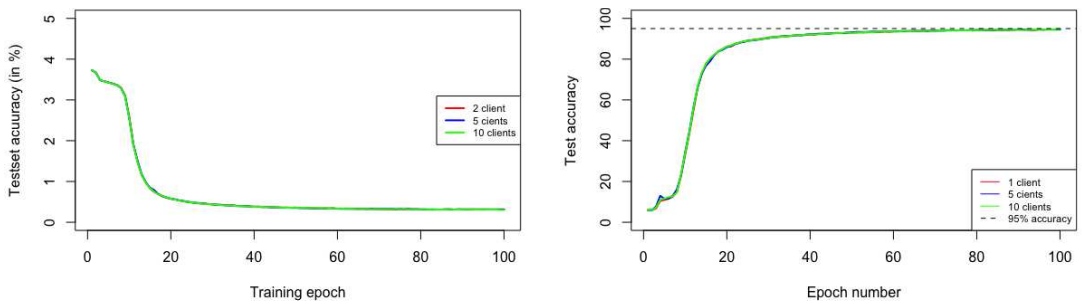
merge strategy	sequential, agregation
number of sources (benign/malicious)	1/0, 2/0, 5/0, 10/0
distribution of the data (non-iid/iid)	iid
backdoor type	none
order of time backdoors are inserted	none
attack model	none
% of poisoned data in malicious nodes	none

4.2 Number of clients

In this section we investigate the effect of a varying numbers of clients. This experiment run is divided into two different sections, as the merging strategy has a significant impact on the results. The configuration for this experiment run is listed in Table 4.1. We test on a variety of numbers of clients and on both merging strategies. Note that the data is distributed independently and identically (iid).

4.2.1 Sequential (cyclic incremental) training

Figure 4.2 shows the relationship between the number of clients and the performance regarding the number of epochs. It can be observed that there is no noteworthy difference in respect to the number of clients in sequential learning.



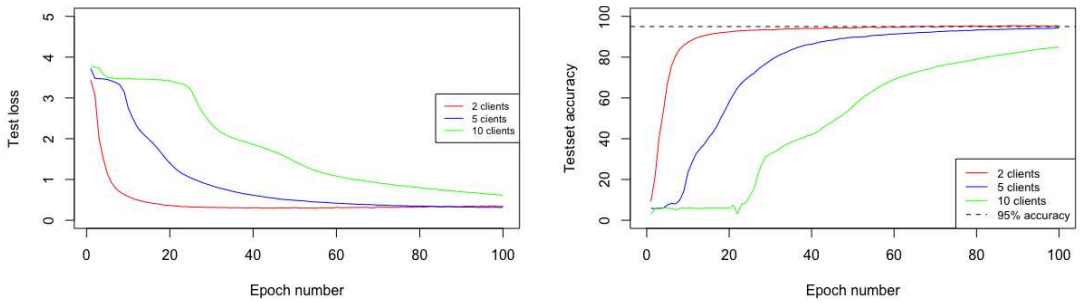
(a) Epoch number vs. test loss (b) Epoch number vs. accuracy on test set

Figure 4.2: Results of number of clients experiments in centralized learning vs. sequential learning

4.2.2 Federated averaging

Our observations show that the more clients are being averaged at one step, the longer the model takes to converge. The overall performance of the global model, however, seems to converge at an identical level.

Graphically, this can be observed in Figure 4.3a as the average test loss decreases slower the more clients are contributing simultaneously. As a consequence thereof, the test set accuracy, depicted in Figure 4.3b, increases slower.



(a) Epoch number vs. test loss (b) Epoch number vs. accuracy on test set

Figure 4.3: Experiments on the number of clients in the case of doing Federated averaging

4.2.3 Comparison of sequential learning and federated averaging

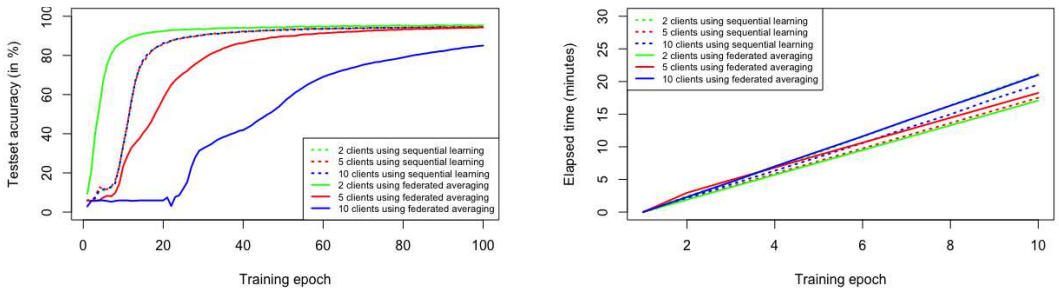
In Figure 4.4 both federated learning methods are compared against each other. Figure 4.4a plots the epoch number against testset accuracy, while Figure 4.4b shows the elapsed time in our testing environment in respect to the number of training epochs.

For a network consisting of 2 clients our observations, depicted by the green lines, show that sequential learning (dotted line) and federated aggregation (continuous line) perform identically after 100 training epochs. Notably, the federated averaging network converges a bit faster.

Testing the same setting for 5 clients using the red lines, one can observe that they again converge to the same level of testset accuracy. In contrast to the previous experiment, this time the sequential learning network converges faster.

In a setting with 10 clients, depicted by the blue line, we see that the sequential learning network again performs the same as it did in smaller sized networks. Federated averaging, however, converges much slower on a higher number of participants, and does not reach the level of the other tested values after 100 epochs. Further tests have shown that a federated aggregation network on this dataset needs more epochs to train, but it eventually reaches the same accuracy as smaller sized networks.

If we consider the amount of time one training epoch needs to finish, we can observe in Figure 4.4b that the differences in time needed per epoch for sequential learning as-well as federated averaging are neglectable. Note: We do not consider communication effects or waiting times which would play a significant role in federated learning. Also, these time measurements are not scientifically reliable - they rather act as a reference point for future measurements. Measuring efficiency in detail remains to be part of future work.



(a) Testset accuracy vs. training epochs (b) Elapsed time (in minutes) vs. training epochs

Figure 4.4: Networks using different numbers of participants in sequential learning and federated averaging

Table 4.2: Configuration used for experiments on the distribution of data

merge strategy	sequential, agregation
number of sources (benign/malicious)	5/0
distribution of the data (non-iid/iid)	non-iid
backdoor type	none
order of time backdoors are inserted	none
attack model	none
% of poisoned data in malicious nodes	none

4.3 Distribution of the data

This section investigates the influence of the distribution of the data. While the other sections' experiments are based on the assumption of all data classes being available on all clients, the influence of some classes being only known to some clients is tested here - in other words: classes being not independent and identically distributed (non-iid).

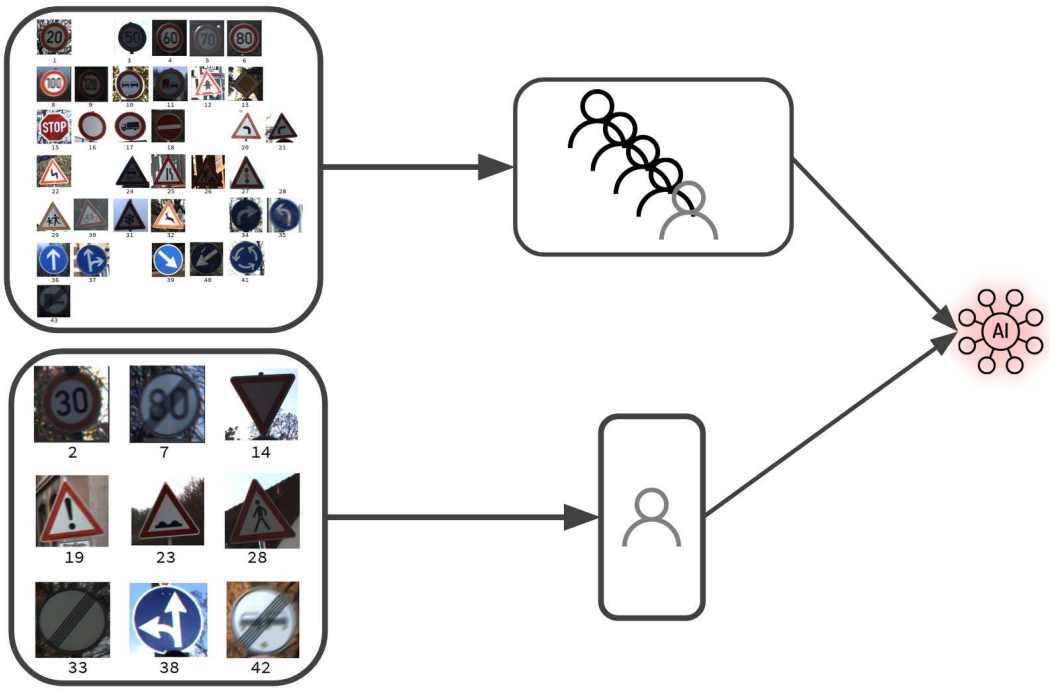


Figure 4.5: Classes split by the ratio of 80:20

For both, sequential learning as well as federated averaging, we have tested two different scenarios.

First, we randomly selected 34 classes, representing roughly 80% of all classes, under the assertion that these 34 classes also make up around 80% of the total number of

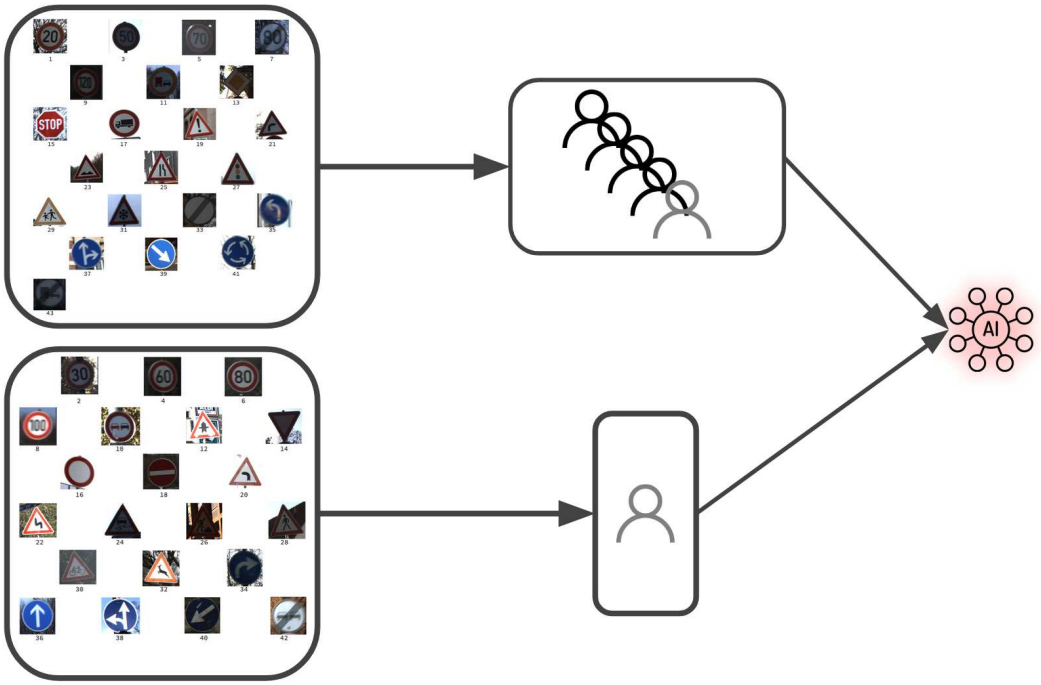


Figure 4.6: Classes split by the ratio of 50:50

observations (denoted as "bigger fragment"). These classes are distributed across all 5 participating nodes. The remaining 9 classes ("smaller fragment"), corresponding to around 20% of the dataset, are distributed only among 1 clients. An illustration of this procedure is depicted in Figure 4.5.

Second, we test the setting of splitting the dataset into 2 parts of around 50% of the whole dataset. We divided the dataset into even and uneven class numbers, and resulted in two fragments. One containing 21 classes (at 4810 observations), and the other containing 22 classes (at 5071 observations). The bigger fragment is distributed across all participants, while the smaller fragment is only known by one (out of five) clients. The precise class splits are depicted in Figure 4.6.

4.3.1 Sequential (cyclic incremental) training

Both split strategies are investigated in the following.

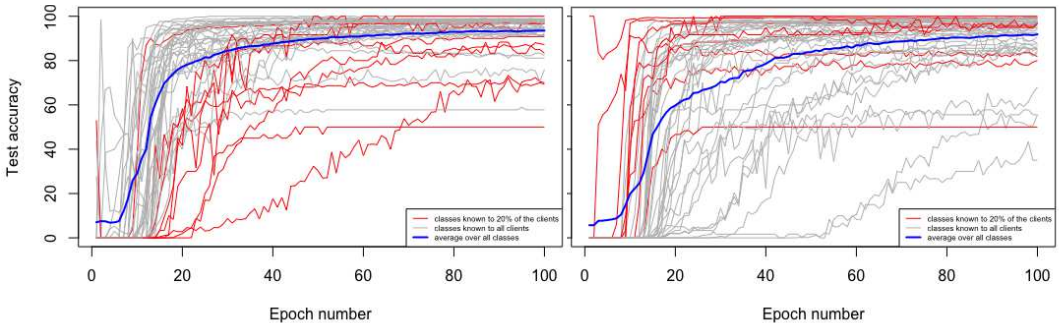
80/20 split
Figure 4.7a shows the results of the smaller fragment of classes being trained before the other classes in each cycle. Each red line represents a class of these "exclusively known classes", and the grey lines represent classes known to all clients. The x-axis represents the number of current train epoch, and the y-axis the per class accuracy on the testset.

If we compare these results to the results where all classes are distributed in an iid way

(see Figure 4.10a), we can clearly observe that the exclusively trained classes are trained slower and, partly, at a lower level of performance. In comparison, classes known to all clients are performing at a comparable level as in the iid case.

In Figure 4.7b the results of training the smaller fragment of classes at the end of each training cycle is depicted. Again, these classes, representing the the exclusively known classes, are marked by the red lines. In general, they are learned earlier than the classes in the bigger fragment. In contrast to the iid scenario depicted in Figure 4.10a, the smaller fragment is performing on a higher accuracy also in the early stages of the model.

In the above tested scenarios, the bigger fragment of data was known by all 5 clients, while 1 client is additionally containing the smaller fragment of classes in their training data.



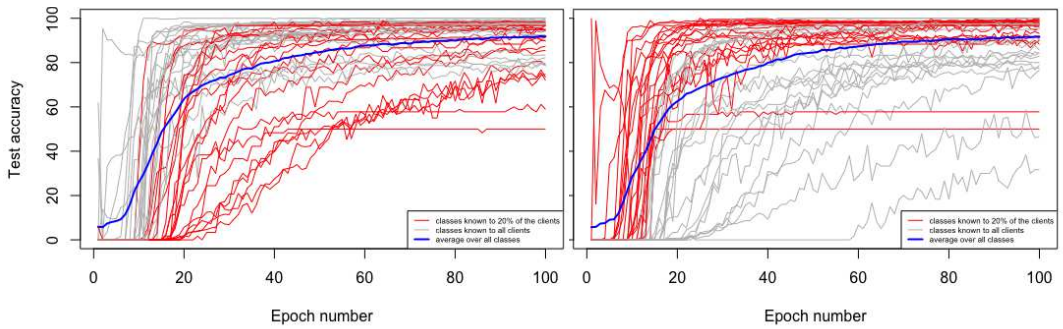
(a) Results of data distribution experiments, the smaller fragment is known by only 20% of the clients using **sequential training**. Exclusively known classes are trained **first**

(b) Results of data distribution experiments, the smaller fragment is known by 20% of the clients using **sequential training**. Exclusively known classes are trained **last**

Figure 4.7: Testing sequential learning for non-iid data using the 80/20 split

50/50 split

Results from splitting the train set into two almost equally sized parts and distributing half of the classes to only 20% of the clients are depicted in Figure 4.8. The outcome confirms the observations made in the first experiment run.



(a) Exclusively known classes are trained **first** (b) Exclusively known classes are trained **last**

Figure 4.8: Testing sequential learning for non-iid data using the 50/50 split

Comparing non-iid results to iid results

In the Figure 4.9 we compare the results of our non-iid testcases with those obtained when the data is distributed identically over 5 clients. This graph shows the per class accuracy when data is distributed iid. Here, the exclusively known classes from our non-iid experiments in this section are marked red to see how they behave in an iid case. Setting: sequential learning, smaller fragment trained last.

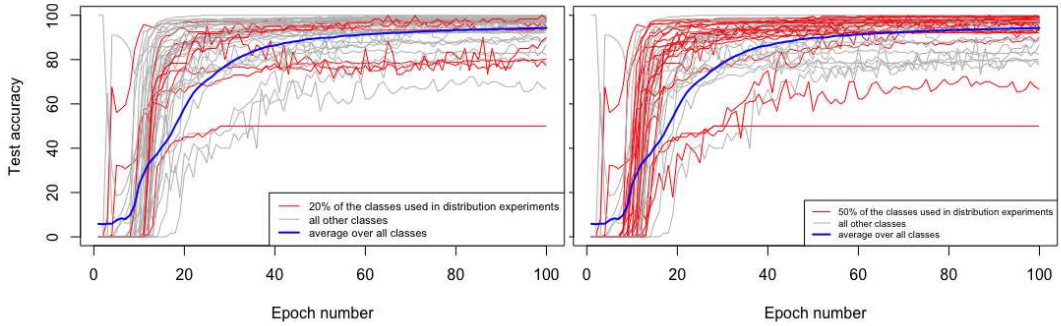
In the 80/20 split and training the smaller fragment after the bigger fragment (Figure 4.7b), we observe a slightly sooner convergence of the same classes of the smaller fragment in comparison to the iid class distribution in Figure 4.9a.

The earlier convergence of the smaller fragment also appears in the 50/50 split (compare non-iid case in Figure 4.8b and Figure 4.9b for the iid case).

For the bigger fragment of classes it can be observed that there is a high variance in their performance in both cases. Meaning, that some classes are still performing similar to the iid case, while some are take longer to converge or do not reach the accuracy after 100 epochs at all.

In Figure 4.9 the overall average accuracy over all classes is depicted. We only see small deviations from the non-iid case to the iid case. Detailed results are depicted in Table 4.3.

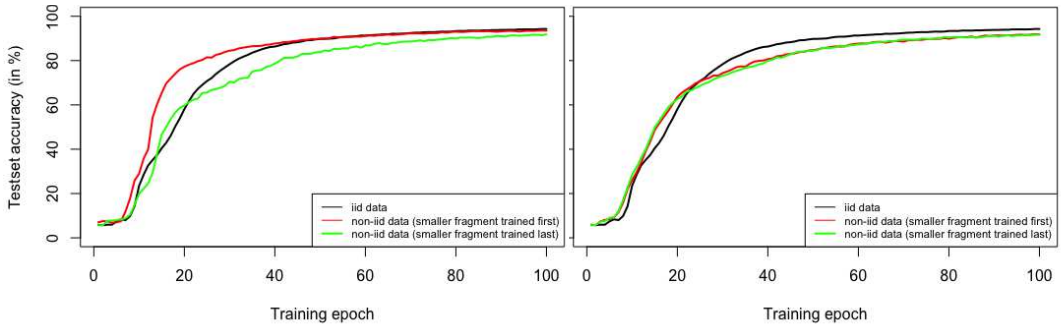
General note: There are some classes that are classified significantly lower level than others. One particular class we investigated is the class with the id 22, that is classified at a accuracy of 57%. A possible reason for this is that in the testset there are many overexposed photos where the red border of the traffic sign is not or hardly visible. In the training set, however, there are no such observations.



(a) Per class accuracies obtained using 80/20 split

(b) Per class accuracies obtained using 50/50 split

Figure 4.9: Comparing the performance of the smaller class fragments to the their respective performance in an iid scenario



(a) Comparing the 80/20 split to results obtained using iid data

(b) Comparing the 50/50 split to results obtained using iid data

Figure 4.10: Comparing the average accuracy over all classes in a non-iid sequential learning scenario with those obtained in iid scenario

4.3.2 Federated averaging

The same scenarios is also tested using federated averaging.

80/20 split

In Figure 4.11 we again mark all "exclusively known classes" red and all commonly known classes grey. We observe that exclusively known classes take way longer to be learned than classes known to all clients, and do not reach the same level of performance. Across all classes, the global model delivers an accuracy of 86%. Apparently some classes (like class ids 33 at a per class accuracy of 66%, id 38 at 88%) perform better than others (id 19 at 7% or id 28 at 0%). After deeper investigation, the assumption that these deviations relate with the number of observations per class have not has turned out to be wrong.

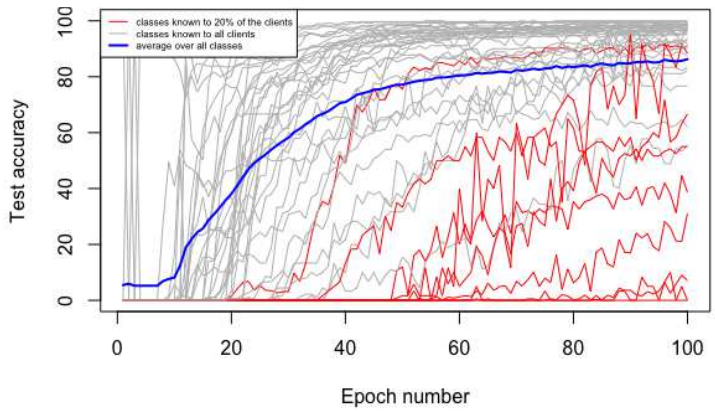


Figure 4.11: Results of data distribution experiments on the 80/20 split using **federated aggregation**

50/50 split

Experiments with 50% of the classes known to 20% of the clients leaded to an even lower accuracy for the exclusively known classes. This results in an overall accuracy of around 50%. The per-class accuracies are depicted in Figure 4.12.

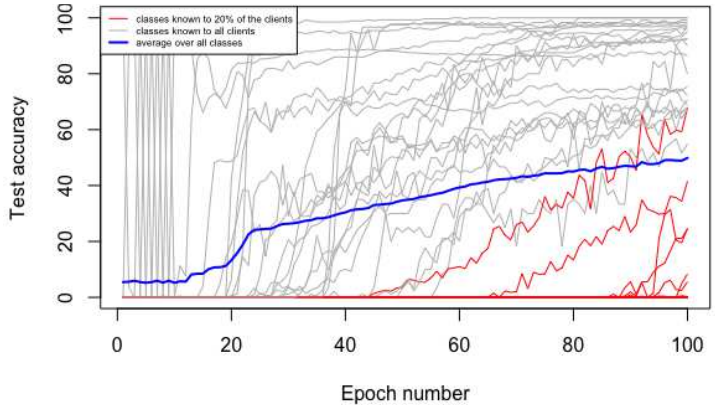


Figure 4.12: Results of data distribution experiments on the 50/50 split using **federated aggregation**

Comparing non-iid results to iid results

In the following, we compare the results of federated averaging with non-iid data using the 80/20 split to those obtained when all data is distributed in an iid way. In Figure 4.13, the per class accuracy of an iid training setting of 5 clients is depicted. The lines that are colored in red represent the same classes that are used in the smaller fraction of the non-iid experiments. This way one can compare their per class accuracies in both cases. In comparison with Figure 4.11, the we see a clear trend that the classes used in the smaller fragment are trained to a lower accuracy. This also effects the overall averaged accuracy over all classes, as represented in Figure 4.14. Throughout the epochs,

the overall accuracy of the non-iid scenario (red line) increases much slower.

This trend increases by looking at the 50/50 scenario, depicted in Figure 4.12. The per class accuracies of the smaller class fragment increase much slower than those in the bigger fragment - at an even lower rate than in the 80/20 scenario. By comparing the average testset accuracy over all classes, we observe that it only reaches 49% after 100 training epochs.

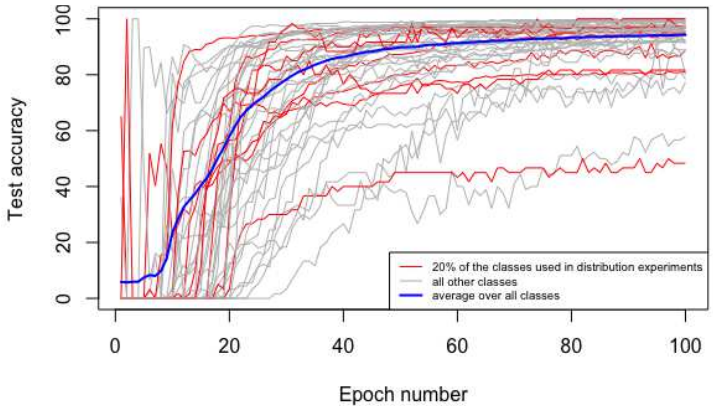


Figure 4.13: Graph showing the per class accuracy when data is distributed iid. Here, the exclusively known classes from our non-iid experiments in this section are marked red to see how they behave in an iid case. Setting: **federated aggregation** and 80/20 split

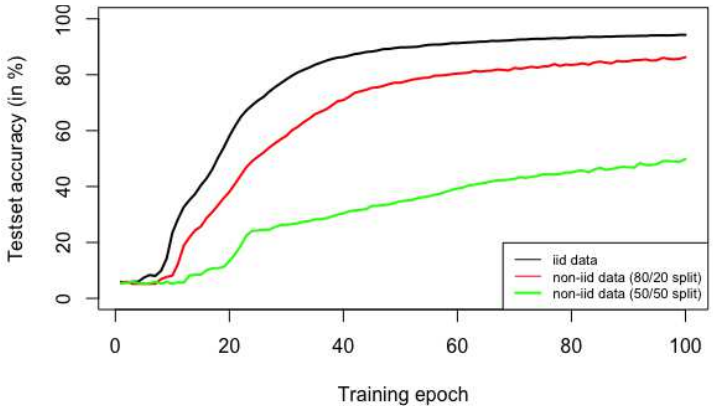


Figure 4.14: Comparing the average accuracy over all classes in a non-iid federated averaging learning scenario with those obtained in iid scenario

Table 4.3: An overview on the overall averaged class accuracies in all tested scenarios

	overall avg. accuracy
iid data - sequential learning	94.75%
non-iid data - sequential learning - 80/20 split - small fragment first	93.49%
non-iid data - sequential learning - 80/20 split - small fragment last	91.82%
non-iid data - sequential learning - 50/50 split - small fragment first	91.80%
non-iid data - sequential learning - 50/50 split - small fragment last	91.66%
iid data - federated averaging	94.26%
non-iid data - federated averaging - 80/20 split	86.27%
non-iid data - federated averaging - 50/50 split	49.87%

4.3.3 Conclusion

In general it can be observed that, according to our experiments, the class distribution does have a significant influence on the federated learning performance. In sequential learning, especially the order of presentation of the classes has a significant relevance. When sequential learning deals with non-iid data, training these samples in the end of a learning cycle indirectly acts as boosting the learning process on these samples, as they are represented to a higher degree in the final model.

In a setting with federated averaging, data that is distributed non-independently and non identically drastically reduces the overall model performance, and the performance of the "exclusive" classes increases later and slower. This effect increases with the number of exclusively known classes.

An overview on the accuracies across all tested scenarios is given in Table 4.3.

Table 4.4: Configuration used for experiments on the order of time the backdoors are inserted

merge strategy	sequential
number of sources (benign/malicious)	4/1, 9/1, 19/1
distribution of the data (non-iid/iid)	iid
backdoor type	green 1% sized squared
order of time backdoors are inserted	backdoors first, backdoors last
attack model	basic
% of poisoned data in malicious nodes	100%

4.4 Timing of the attack (in sequential learning)

In sequential learning, backdoors can be introduced at different points of time. As we have seen in literature, an intrinsic property of sequential learning is catastrophic forgetting [KPR+17], meaning that data that is learned in early stages of the learning cycle is likely to have less influence at the end of the cycle (in other words, being "forgotten"). A detailed description of this phenomenon is also given in Chapter 6.

The experiments in this section give an insight on the behaviour of the model when the poisonous data is inserted at the beginning and the end of the learning cycle. The two different runs are illustrated in Figure 4.15. "Attacker first" means that the backdoored clients are trained **before** the benign clients at the beginning of each training epoch. In contrast, "attacker last" means that the malicious clients are always trained **after** the benign clients at the end of the cycle. We perform these tests at different sized networks, and assume that each client has the same amount of data. Following Section 4.5 we choose to use the green 1% sized backdoor for these experiments, as this backdoor performs best.

The detailed configuration is listed at Table 4.4.

(a) Attacker first (b) Attacker last

Figure 4.15: Illustrations of attacker participating at different stages of the learning cycle in a sequential network (non-poisoned models red, poisoned model green)

We perform testset evaluations twice per learning cycle - the first evaluation is done after training all benign clients, while the second is performed after the adversary is trained (in other words, when the learning cycle is finished).

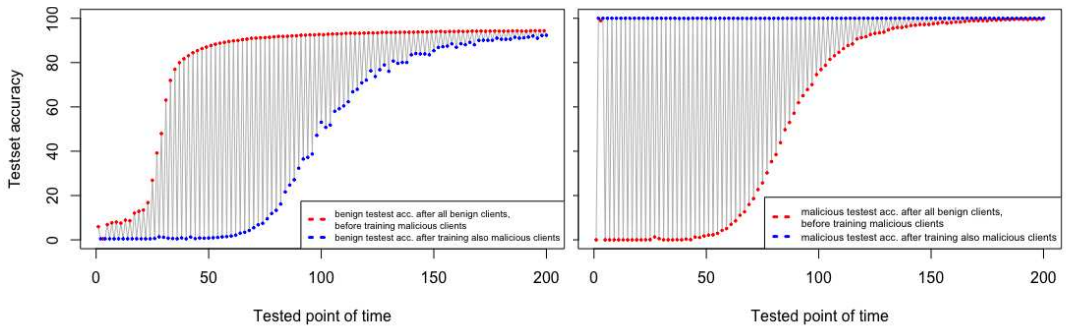
In the following Figures 4.16-4.21 we observe the results on benign and malicious testsets

on the above mentioned experiment configuration. The abscissas of these plots represent the "tested point of time". Here, one point of time equals one evaluation on the test set - testing two times per epoch results in 200 point of test times. The ordinate represents the accuracy on the testset. Red dots marked on the graph represent tests after the first set of clients are trained, but before second set of clients is trained. Blue dots represent evaluations after all clients are trained and the learning cycle is completed. The grey line represents the curve connecting tested points of time with accuracy.

Figure 4.16 illustrates the result of a federated network consisting of one attacker and 4 benign clients. As always we assume that each client uses the same amount of data for the model training. Therefore, there is in total four times more benign data than poisoned data. In this testcase the four clients, only containing benign data, are trained at the beginning sequentially. After they finished training, the adversary is trained, and the first learning cycle is finished. This procedure is repeated for 100 iterations.

The performances on the benign testset (Figure 4.16a) shows clear differences in the performance between the results after the four benign clients and the performance after the last (malicious) client is trained. While tested in between the benign and malicious clients, the performance on the benign testset reaches a level of more than 80 percent after around 20 epochs (40 ticks). However, it takes around 75 epochs (150 ticks) to reach this level when a malicious client is trained afterwards.

On the malicous testset (see Figure 4.16b) we can see that the backdoor is introduced immediately after the malicious client is trained, but it takes more than 50 epochs to stay at a level of 80% accuracy after the model is retrained by the benign clients' data. Early trained data seems to be forgotten, especially in the first half of all training epochs.

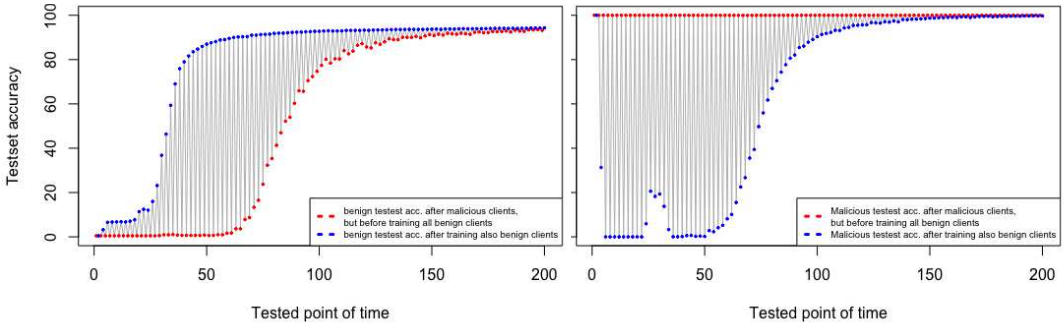


(a) Benign testset (b) Malicious testset

Figure 4.16: Performance of one malicious client trained **after** 4 benign clients

In Figure 4.17 we test the exact opposite case. This time, the malicious client is trained first, followed by four benign clients. Tested after one epoch is finished entirely, we observe that the benign testset takes around 20 epochs (40 ticks) to reach an accuracy of 80%, while the backdoor takes around 45 epochs to be introduced. When tested in between, the backdoor is introduced almost immediately, while the benign data takes more than

50 epochs to reach the 80% accuracy level. The results again show the occurrence of catastrophic forgetting.



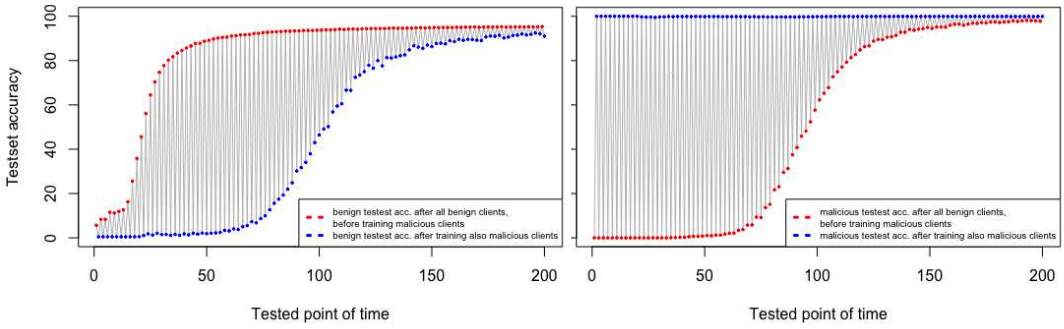
(a) Benign testset (b) Malicious testset

Figure 4.17: Performance of one malicious client trained **before** 4 benign clients

Comparing Figures 4.16 and 4.17, we can see that they are running almost an inverse of each other. When the adversary is trained before the benign clients, the performance on the benign testset increases much faster than the accuracy on the malicious testset. When the adversary is trained after the benign clients, the testset convergence time of the testset performances are the other way around.

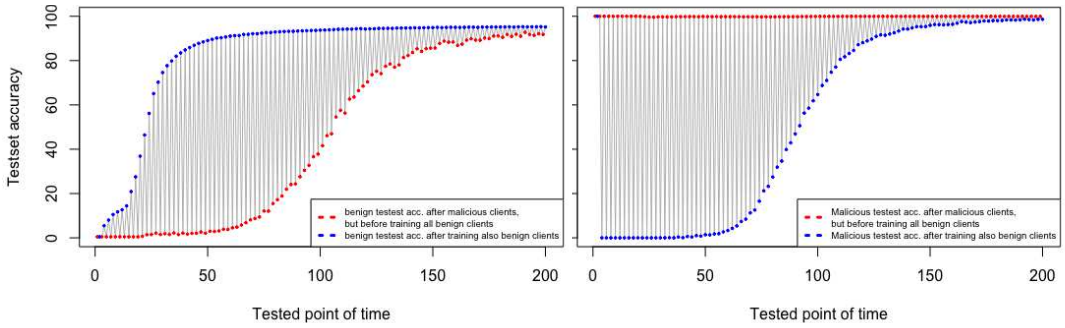
Generally speaking, when backdoors are inserted at the beginning of a learning cycle, the model works well on benign test data way earlier than if they were trained at the end. On malicious data however, this behaviour is completely inverted.

In Figures 4.18 and 4.19 we test a network with nine benign clients and one malicious clients. The results appear to confirm the behaviour observed on the smaller network. This is also the case for the tested network with 19 benign clients and one malicious client depicted in figures 4.20 and 4.21.



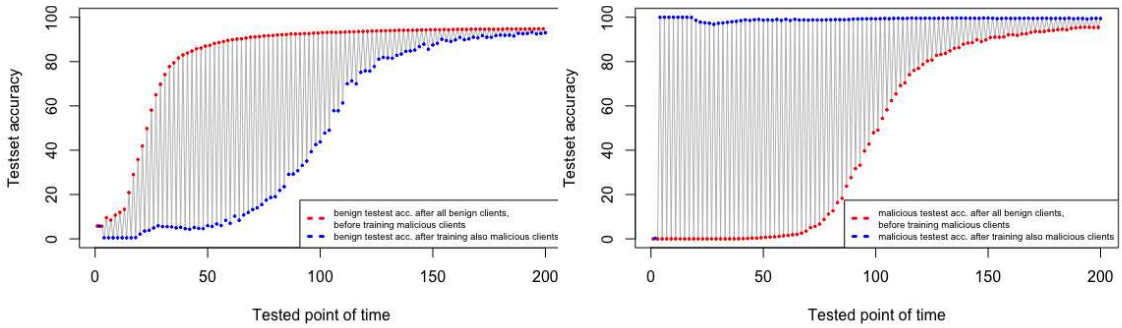
(a) Benign testset (b) Malicious testset

Figure 4.18: Performance of one malicious client trained **after** 9 benign clients



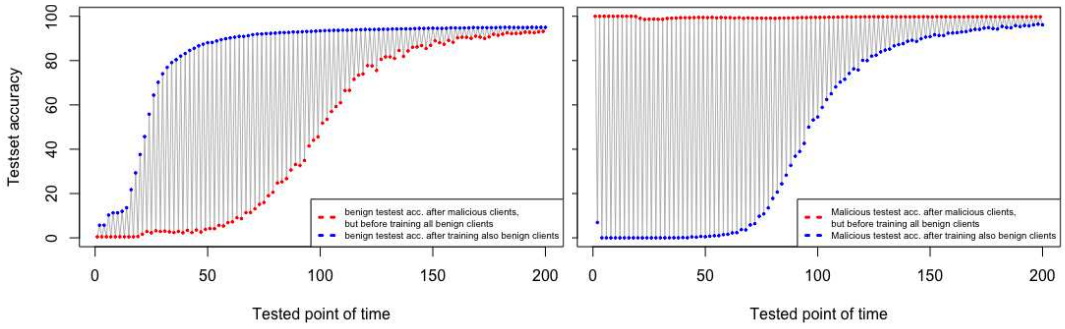
(a) Benign testset (b) Malicious testset

Figure 4.19: Performance of one malicious client trained **before** 9 benign clients



(a) Benign testset (b) Malicious testset

Figure 4.20: Performance of one malicious client trained **after** 19 benign clients



(a) Benign testset (b) Malicious testset

Figure 4.21: Performance of one malicious client trained **before** 19 benign clients

4.4.1 Conclusion

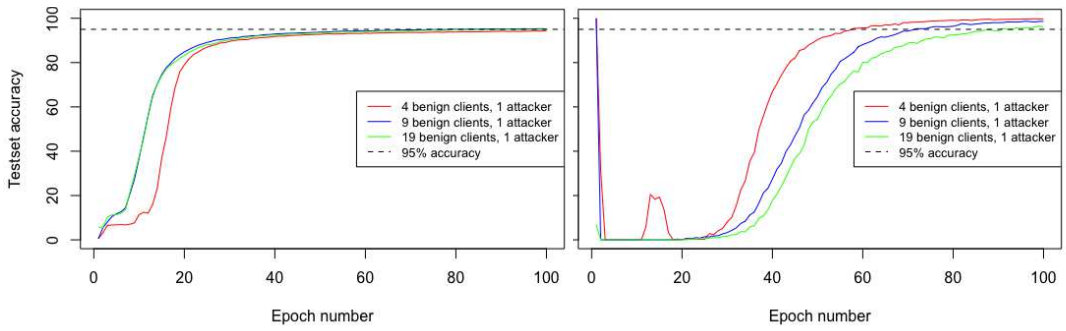
Experiments of this section result that observations trained in later stages of a training cycle have a bigger influence on the model. This is especially true in early training epochs, as we can see in the graphs - here, the "mid-cycle" (first tick) and "end-cycle" (second tick) results differ greatly from each other. This effect decreases with an increasing number of training epochs as these differences begin to disappear.

A detailed comparison of the different sized networks' testset performances is given in Figure 4.22 (adversary trained before benign clients). It turns out that for the benign clients dataset, the networks of 10 and 20 clients perform nearly identically, while at 5 clients it takes 10 epochs longer to reach a converging state. On the malicious dataset, having only 4 benign clients trained after the malicious clients leads to the fastest introduction of the backdoor pattern. But also networks at 9 and 19 benign clients were able to introduce the backdoor, at a delay of 2 respectively 10 epochs.

In Figure 4.23 the same comparison is made for the adversaries trained after the benign clients. In this case the performances are nearly identical for the benign and the malicious testset on both test-sets.

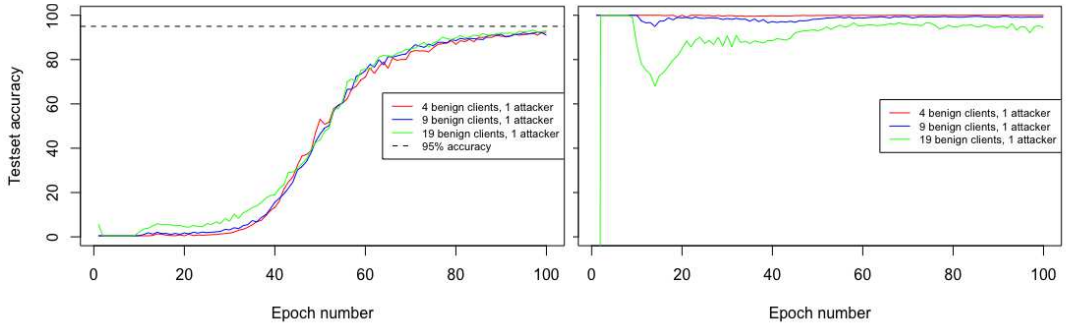
Figure 4.24 directly compares injecting the backdoor in the beginning and at the end (at 4 benign and 1 malicious client). The adversary is trained once in the beginning (red line) and once at the end of the cycle (blue line). We observe that the later an attacker injects the poisoned data each cycle, the earlier it is introduced into the global model. However, the performance on the benign testdata needs the entire period of training to reach an accuracy comparable to a network without adversaries.

These observations lead to the following final conclusions. If the training phase is long enough, the sequence of the participants does not matter. On the other hand, if the training phase is short, the sequence of the clients is very important for an attacker, as it's influence increases. This behaviour could also be helpful for developing a defensive strategy against backdoors in sequential learning. By placing some trusted notes at the end of the cycle and stopping the training at a point when the overall training accuracy converges, one might prevent adversaries from entering the backdoor. More evidence on this strategy is left for future work.



(a) Benign testset (b) Malicious testset

Figure 4.22: Performance of the malicious testset on varying numbers of clients after finishing one epoch - malicious client is trained **before** the benign clients



(a) Benign testset (b) Malicious testset

Figure 4.23: Performance of the malicious testset on varying numbers of clients after finishing one epoch - malicious client is trained **after** the benign clients

Note: During our experiments for this section we observed that the batch size used for training the neural network has a significant impact on the amount of catastrophic forgetting. If we keep the amount of data constant while decreasing the batch size, our data is distributed across a higher number of batches. After processing each batch, the loss is calculated and the model weights are changed. If the batch size is lowered, than more batches have to be trained consecutively, and the more often the model is updated. This increases the effect of catastrophic forgetting, especially if many batches containing only backdoor data are trained in succession. In the following graphic (Figure 4.25) we lowered the batch size from 512 to 128, and use a sequential learning setting of 4 benign clients and 1 adversary trained at the beginning of the learning cycle. We can clearly see that the graphs are fluctuating to at a greater amplitude (in the benign testset for the first 50 epochs between 0% and 95%) compared to Figure 4.17, agreeing with above mentioned statement.

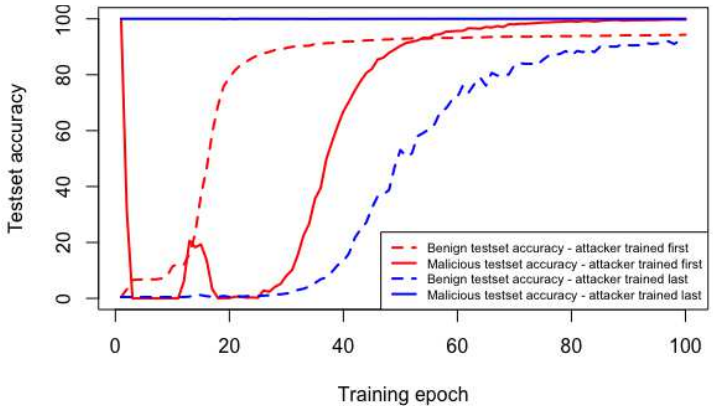
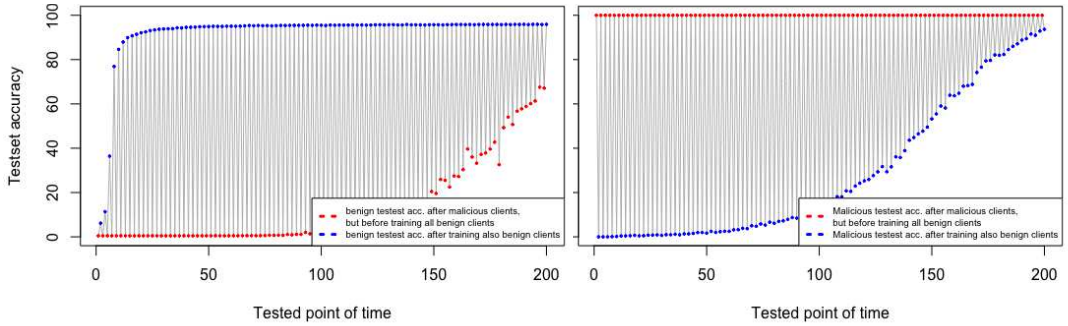


Figure 4.24: Poisoning sequential learning at 4 benign clients and one attacker



(a) Benign testset (b) Malicious testset

Figure 4.25: Performance of one malicious client trained **before** 4 benign clients at a lower batch size (128 instead of 512)

Table 4.5: Configuration used for experiments on the prominence of the backdoors

merge strategy	aggregation
number of sources (benign/malicious)	4/1
distribution of the data (non-iid/iid)	iid
backdoor type	green 1%, black 1%, green 0.5%
order of time backdoors are inserted	none
attack model	model replacement
% of poisoned data in malicious nodes	12.5%, 25%, 50%, 75%, 100%

4.5 Prominence of the backdoor

This section deals with the prominence of backdoor patterns of an attacker. We test different backdoor sizes and colors specific for this dataset to gain an insight on their impact. Tested patterns are depicted in Figure 4.26, and the configuration used for all experiments in Table 4.5.

(a) Green backdoor at the size of ~1% of the image

(b) Green backdoor at the size of ~0.5% of the image

(c) Black backdoor at the size of ~1% of the image

Figure 4.26: Three different backdoors used for experiments on the traffic sign dataset

Following prior work [WYS+19] [GDG17], we also decide to stick with square sized patterns, and also start with patterns making up around 1% of the total image's area. We choose to use a green squared backdoor of 1% size as green is seldom present on traffic signs. For comparison, a color that frequently appears on traffic signs is black. Therefore, we also experiment with 1% sized black patterns. The third tested backdoor, a green squared 0.5% sized pattern is tested to gain an insight on the importance of the backdoor's size on this dataset. The backdoor patterns are located randomly around the center of the image in an interval of [-20,20] percent in x and y direction. In this section we considered a backdoor to be successfully introduced if the performance on the malicious testset exceeds an accuracy of 95%.

We showcase the prominence of different backdoors in a federated aggregation network consisting of 4 benign and 1 malicious client. Each client contains the same amount of data. The merge strategy used is the model replacement strategy, as we were not able to sufficiently enter backdoor pattern using the basic attack strategy (see Section 4.6).

In Figure 4.27, the results using the 1% sized green squared backdoor are displayed. We

can observe that in a network with malicious clients using 25% and 50% backdoored data the performance on the benign as well as the malicious testset exceeds the 95% accuracy threshold after 100 epochs. In this cases we follow that the backdoor attacks were successful.

Comparing these results to the smaller 0.5% green sized backdoor in Figure 4.28, we can clearly see a difference. Backdoor insertion at 25% and 50% malicious data in malicious clients only reach 61% and 1% accuracy on the malicious testset and can be considered as not successful. If 75% malicious data is used in the fraudulent clients, the backdoor is successfully introduced, then however the performance on the benign testset is reduces to 43%.

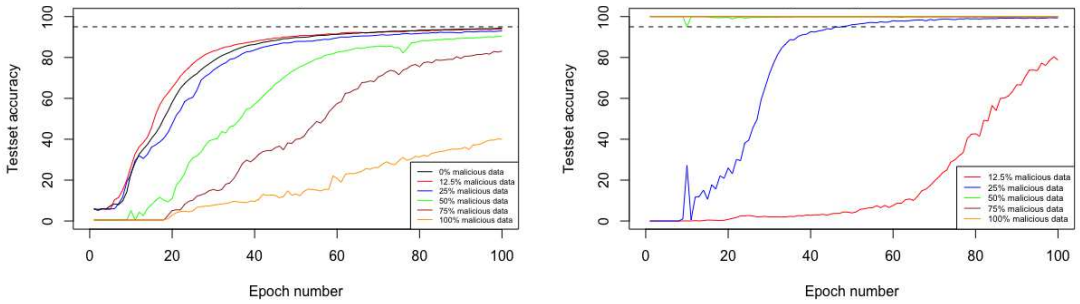
The results of using a black and 1% sized backdoor pattern are displayed in Figure 4.29. In contrast to the 1% sized green squared backdoor, the only backdoor attack considered as successful is when malicious clients use 50% poisoned data. If we use only 25%, the performance of the malicious testset is low, and the attack is not successful.

4.5.1 Conclusion

To summarize, we can definitely see the impact of the pattern's size as well as color on the success rate. In our dataset, bigger backdoors work better than the smaller backdoors, and the (rarely in the images occurring) green color works better than the (very common) black color.

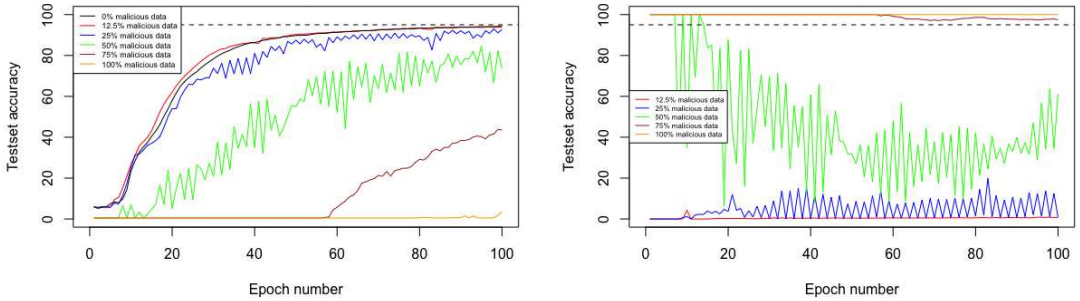
An important factor for the attacker is clearly the inconspicuousness of the backdoor pattern itself. In a real world setting, a backdoor that is highly suspicious will more likely be identified that one that is not, as described in Section 3.5. While there is no quantitative metric of what "looks suspicious", one can generally argue that the size plays a significant role. When investigating real world traffic signs, as well as signs available in the German Traffic Sign Recognition Benchmark dataset itself we found many stickers exceeding this 1% area by far. Also, they occur in a range of colors and illustrate a variety of motives. Examples from the mentioned dataset are shown in Figures 4.30.

Recommendations for an attacker are, according to above observations, to select a big and unusually colored backdoor pattern to increase the chance of the attack to be successful. More backdoor patterns and comparisons to existing literature are discussed in Chapter 6.



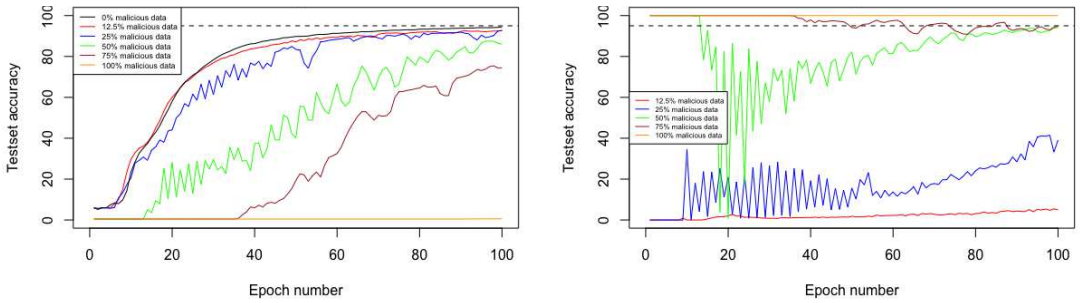
(a) Benign testset (b) Malicious testset

Figure 4.27: Results of backdoor insertion results using model replacement method on **1% sized green squared** backdoors on different percentages of poisoned data in malicious clients



(a) Benign testset (b) Malicious testset

Figure 4.28: Results of backdoor insertion results using model replacement method on **0.5% sized green squared** backdoors on different percentages of poisoned data in malicious clients



(a) Benign testset (b) Malicious testset

Figure 4.29: Results of backdoor insertion results using model replacement method on **1% sized black squared** backdoors on different percentages of poisoned data in malicious clients

Figure 4.30: Naturally occuring backdoors in samples of the German Traffic Sign Recognition Benchmark dataset

Table 4.6: Configuration used for experiments on the attack model

merge strategy	aggregation
number of sources (benign/malicious)	4/1
distribution of the data (non-iid/iid)	iid
backdoor type	green 1% sized squared
order of time backdoors are inserted	none
attack model	basic, model replacement
% of poisoned data in malicious nodes	12.5%, 25%, 50%, 75%, 100%

4.6 Attack Strategies (in federated aggregation learning)

This section deals with two different attack strategies as described in Section 2.6.1. These strategies are compared in a federated learning setting using federated averaging, as the model replacement strategy is not capable of used in a sequential learning setting. The backdoor used in this section is the "green 1% sized backdoor", as it is the best performing pattern according to Section 4.5. In this section we give a performance comparison on both methods, including an analysis on the fraction of benign to malicious samples in malicious clients.

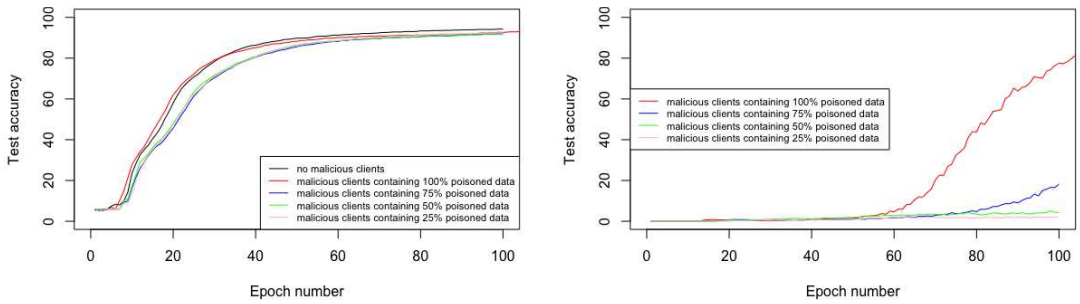
The experiment's configurations are listed in Table 4.6.

4.6.1 Basic strategy

The following tests in the basic attack strategy scenario are done in a network consisting of 5 clients in total. We choose to use a 4/1 split (4 benign and 1 malicious client) as it is the configuration with the best success rate for the attacker. If the attack does not work in this setting, it will be even less performant if the number of attacks in relation to benign clients is decreased.

In Figure 4.31 we observe that the backdoor insertion into the global model using the basic attack strategy performs best when all available data in the malicious clients is poisoned. As we can see, in the case of 100% poisoned data in the clients, the performance climbs to a level of around 80% after 100 epochs. The first 60 epochs this accuracy is near to 0%. The success rate of the attack drastically reduces if the number of clients in a federated network increases (compare Figure 4.37). When increasing the maximum epoch numbers to 200, we observe that the accuracy on the malicious testset steadily increases. It eventually exceeds the 95% testset accuracy after 145 train epoch for malicious clients containing 100% poisonous data.

In even bigger neural networks, accuracy of this strategy on the malicious testset decreases drastically (under 1%, see Section 4.7.2), making it unusable for backdoor insertion.



(a) Benign testdata (b) Malicious testdata

Figure 4.31: Basic attack strategy in federated aggregation at different percentage of poisoned data in malicious clients

4.6.2 Model replacement strategy

For the model replacement strategy we observe that an elementary parameter of importance is the fraction p, defined as the relation between benign and malicious samples in malicious clients. The reason for its high importance is that the goal of this methodology is to replace the global model with the attacker's local model - and to be also performing successfully on benign training data, the malicious clients need to have correctly labeled observations as well.

In Figure 4.27 we test values for p between 100% (only poisoned samples in the attackers train set) and 0% (no poisoned samples). It turns out that this fraction acts a a trade-off between performance on the benign testdata and performance on the malicious testdata. For our use-case the optimal value for p appears to lie between 25 and 50%. These observations agree with findings in literature as described in more detail in the Evaluation-Chapter 6.5. Bagdasaryan et. al. [BVH+18] for example used a percentage of 31% at 5% poisoned clients to reach an accuracy on their malicious testset of over 90%.

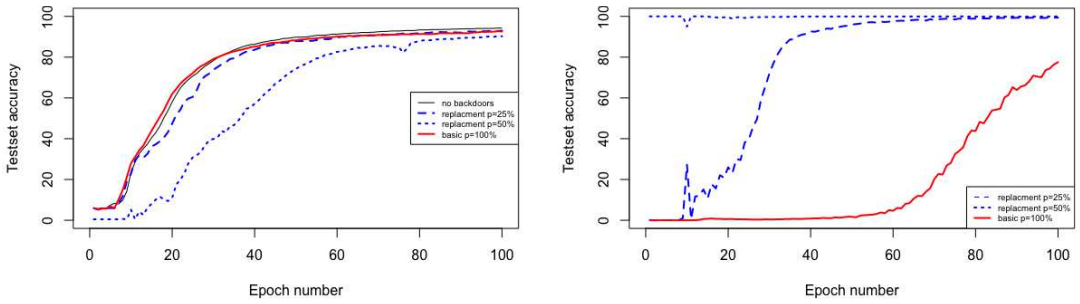
An interesting finding is that even at p=100% (containing no correctly labelled data), the global model is still able to classify some benign testsamples correctly. This means that there is only one (wrongly labeled) class in the train set of the adversary. A plausible explanation for this behaviour is that the global model is not entirely replaced by the local model though replaced to a high degree.

4.6.3 Comparison

Comparing these two strategies, we can see that the model replacement strategy clearly outperforms the basic attack strategy by all means. Our best performing value in the basic attack strategy (p=100%) reaches a lower level of backdoor accuracy compared to the best performing value using model replacement strategy, and also needs way more training epochs to be inserted (see Figure 4.32).

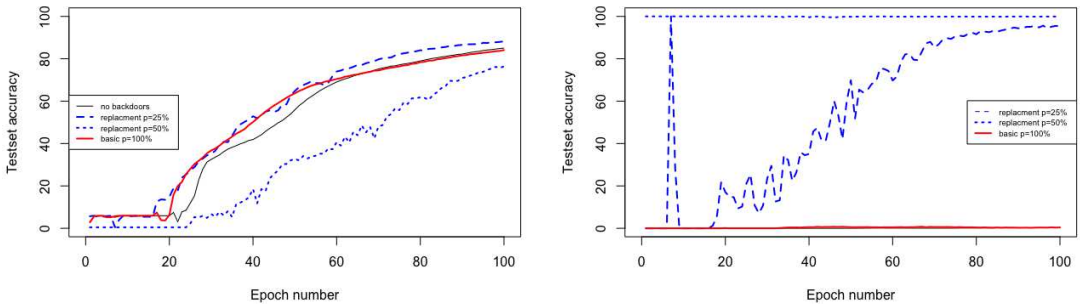
Furthermore, as we can see in Section 4.7, the more clients are averaged at one point

of time, the bigger this performance difference between the two methods becomes. The same comparison as above is now tested in a network consisting of 8 benign clients and 2 malicious clients. In Figure 4.33 this comparison is visualized graphically. The performance difference of the basic attack strategy and the replacement strategy has increased.



(a) Benign testdata (b) Malicious testdata

Figure 4.32: Comparing basic attack strategy to model replacment strategy in a network consisting of **4 benign clients and 1 malicious** clients



(a) Benign testdata (b) Malicious testdata

Figure 4.33: Comparing basic attack strategy to model replacment strategy in a network consisting of **8 benign clients and 2 malicious** clients

Table 4.7: Configuration used for experiments on the number of attackers

merge strategy	sequential, aggregation
number of sources (benign/malicious)	4/1, 3/2, 2/3, 9/1, 8/2, 7/3
distribution of the data (non-iid/iid)	iid
backdoor type	green 1% sized squared
order of time backdoors are inserted	last
attack model	basic, model replacement
% of poisoned data in malicious nodes	12.5%, 25%, 50%, 75%, 100%

4.7 Number of attackers in relation to benign clients

In this section we experiment on the number of benign clients in relation to the number of malicious clients, and their influence on the global model's accuracy. We again use a green squared 1% sized backdoors as it appears to be working best. The detailed experimental setup in Table 4.7 is used for the experiments in the section.

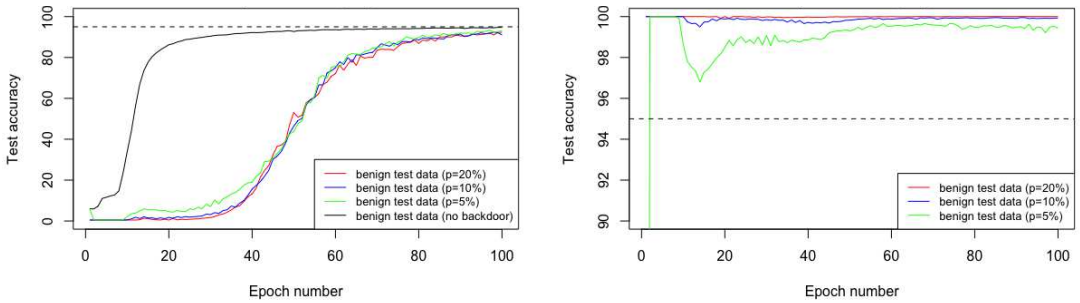
Again, we divide this section into sequential training and federated averaging, as this makes a difference in the results.

4.7.1 Sequential (cyclic incremental) training

Similar to experiments in literature summarized in Section 2.6, we test values for the fraction of attackers between p=5% and p=20%.

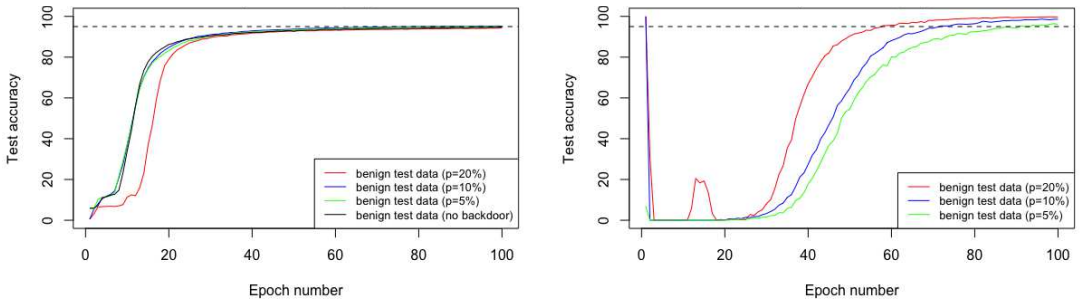
In Figure 4.34 we show the results when backdoored clients are trained after all benign clients. We can conclude that all tested values lead to an accuracy of over 95% on test-data containing backdoor images, while also achieving an accuracy of over 95% on the benign test data, after 100 training epochs. While the performance on the benign testset increases notably slower than in a network without fraudulent clients, the backdoor is entered immediately in all tested cases.

In Figure 4.35 we perform the same experiments as done for Figure 4.34 but train the malicious clients first. Also in this case, all tested values reach the 95% performance level on both testsets. It appears that in our case study the number of clients in sequential learning is hardly relevant.



(a) Benign testset (b) Malicious testset

Figure 4.34: Testset performance with p% of backdoor clients using sequential training, training backdoor clients last



(a) Benign testset (b) Malicious testset

Figure 4.35: Testset performance with p% of backdoor clients using sequential training, training backdoor clients first

4.7.2 Federated averaging

We divide this section into two parts, corresponding to the two attack strategies mentioned in Section 2.6.1.

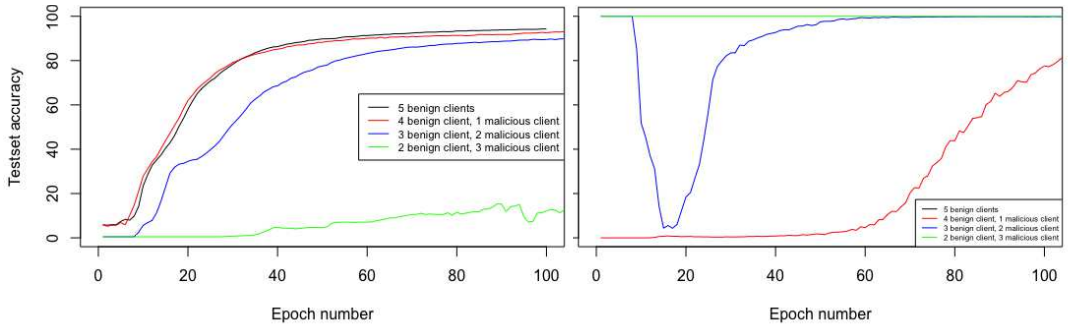
Basic attack strategy

As mentioned in Section 4.6, the best results using basic attack strategy were achieved with clients containing 100% malicious data. Therefore, we use this value for all further tests in this section. Also, we examine two different sizes of federated networks.

First, we test the basic attack strategy in networks with 5 clients. We test having one malicious client (20% of the network, represented by the "red line" in figures), two malicious clients (40% and the "blue line" in figures) or three malicious clients (60%, displayed by the "green line"), while the rest are benign clients. The results are depicted in Figure 4.36.

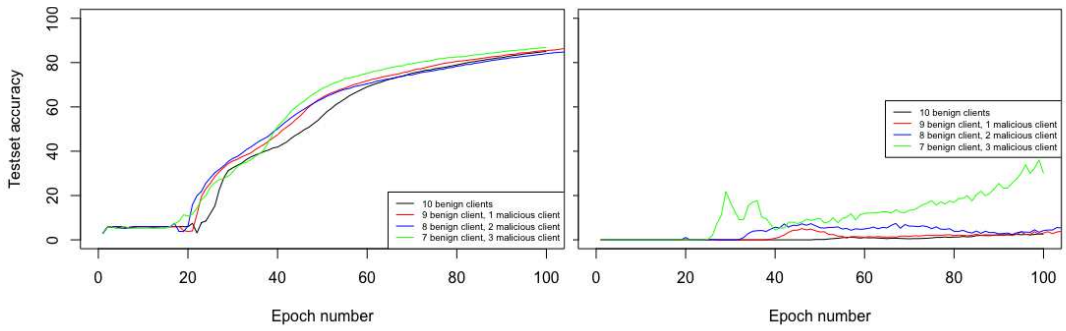
Also, a network with 10 clients is examined and tested for the influence if the number of malicious clients is increased to 1 (10%), 2 (20%) or 3 (30%). Figure 4.37 shows the results.

Our experiments conclude that a higher percentage of malicious clients in respect to benign clients leads to a better insertion of the backdoor into the global model. However, a factor that is surprisingly more important is the absolute number of clients averaged over one epoch. Comparing a malicious client rate of 20% we observe that in a network consisting of 5 clients (Figure 4.36) the backdoor is introduced at a much higher level than in a network of 10 clients (Figure 4.37).



(a) Benign testset (b) Malicious testset

Figure 4.36: Varying the percentage of malicious clients in a network of 5 clients with basic attack strategy



(a) Benign testset (b) Malicious testset

Figure 4.37: Varying the percentage of malicious clients in a network of 10 clients with basic attack strategy

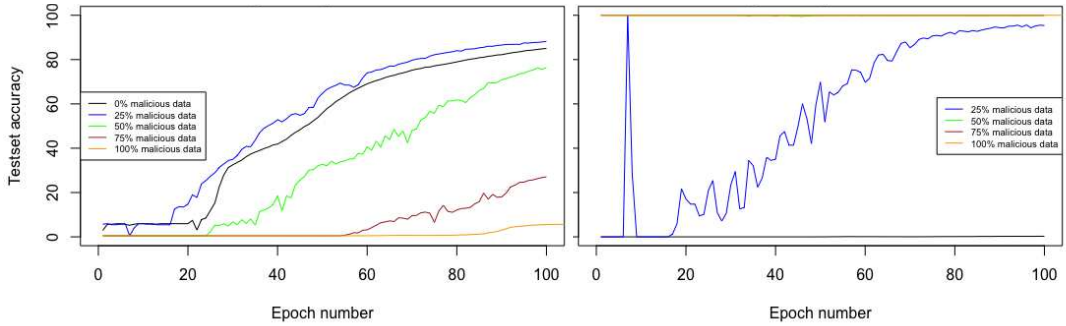
Model replacement strategy

The goal of the model replacement strategy is that the attacker replaces the global model. If more than one attacker is present in the network, they collaborate in the way that the attackers split their poisoned model into n-parts, and each attacker sends $1/n$ for aggregation.

In Figure 4.27 we can observe results from experiments on 4 benign clients and 1 malicious client, containing 20% malicious clients. To keep comparability with prior experiments, we repeated the exact same experiment configuration in a network of 9 benign and 1 malicious client, resulting in 10% malicious clients. These results are depicted in figure 4.39. Each graph again shows performance curves at different malicious clients poison rate p.

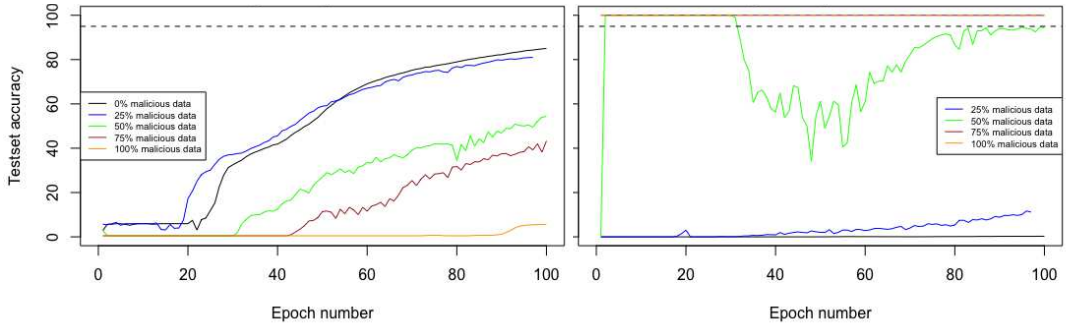
In both cases, the backdoor is successfully introduced at a performance of >95% in the case of p>50%. In the case of 10% malicious clients, even p=25% is enough to reach this performance threshold.

For the performance of the benign testdata, we observe some differences. In general, networks with a higher number of participants take longer to converge than networks with fewer participants (see also Section 4.2 for discussion). Noteworthy, the benign testset accuracy of our network at 20% malicious clients (Figure 4.38) increases faster for p=50% and p=75% than our network at 10% malicious clients (Figure 4.39).



(a) Benign testset (b) Malicious testset

Figure 4.38: Experiment scenario from Figure 4.27 at 8 benign clients and 2 malicious clients



(a) Benign testset (b) Malicious testset

Figure 4.39: Experiment scenario from Figure 4.27 at 9 benign clients and 1 malicious client

4.7.3 Conclusion

To summarize, in the setting of sequential learning, the effect of the percentage of malicious clients is relatively small. In all tested settings for malicious clients fractions in the range of 5% to 20% we were able to introduce the backdoor into the model.

In comparison, in a network using federated aggregation this effect is much bigger. Also, our experiments concluded that the fraction of malicious clients has a significant impact on the speed of embedding for the backdoor - for the basic attack strategy and model replacement strategy. In general, a higher number of attackers in relation to benign clients has a positive influence on the time the backdoor needs to be introduced into the global model.

Notably, similar to observations by Nguyen et. al. [NRMS20] the effectiveness of a backdoor pattern is not simply connected to the amount of poisoned data in the network. Instead, the way the poisoned data is distributed among adversaries is a highly important factor.

4.8 Analysis

In this case study we investigated on the behaviour of federated learning in terms of a varying number of clients and its performance on different types of data distributions (non-iid and iid). We were discussing differences in sequential learning and federated averaging architectures, as well as the possibility of backdoor attacks in the setting of traffic sign classification. To gain an insight, we have implemented a state of the art neural network that is used on the German Traffic Sign Benchmarks Dataset. Additionally, we altered this dataset and added three types of backdoor patterns to the observations - at varying colors and sizes. Using this environment we performed a range of attacks using different parameters on both types.

We show that in the case of sequential learning, the number of clients regarding effectiveness of the model is not important if the data is distributed independent and identically. However, due to communication effects between the distributed clients there is definitely an impact on the efficiency, but we have not put our focus on that. Furthermore, observations from the early stages of a sequential learning cycle are more likely to be forgotten at the end, a behaviour denoted as "catastrophic forgetting" in literature. For the federated averaging case, we observe that the more clients are averaged simultaneously, the longer the global model needs to converge, but they eventually reach the same level of effectiveness. Furthermore, we show that both techniques have problems with non-iid data, and in the federated aggregation case, sparsely known data is trained slower.

Addressing backdoor attacks, we showed that a state of the art neural network used in this domain is very vulnerable - in both, sequential learning and federated aggregation settings. Bigger-sized and seldom occurring colored backdoor patterns (in respect natural occurring patterns in the benign training data) show a better performance on our dataset, as well as a higher percentage of malicious clients. Backdoors can introduced by both tested attack strategies, whereas the model replacement strategy clearly outperforms the basic attack strategy. To achieve best results with the latter strategy, an attacker must tweak the fraction between benign and malicious observations in his data. This fraction acts as a trade-off between performance on the benign data and the backdoor performance.

In the next chapter, we further investigate backdoor related research questions in the domain of facial recognition. Subsequently, we evaluate all our results against observations obtained in existing literature.

CHAPTER 5

Case study on face recognition

In this chapter we investigate backdoor attacks in the context of face recognition. We use the Yale Faces dataset and add backdoor-patterns that could be applied in reality. In contrast to the previous chapter we primarily focus here on the effects of the different backdoor patterns.

For this dataset we use a neural network architecture which is described in more detail in Chapter 3.3. The optimizer used for this network is Adam [KB15] at a learning rate of 0.0001. The reason we choose Adam is that this optimizer performed well on face recognition tasks in the past [AMDJ12]. The value of 0.0001 is the default learning rate set in the PyTorch framework, which clearly outperformed also tested values of 0.001 and 0.00001.

As explained in Chapter 3 we use two different kinds of backdoor patterns: a "full beard" pattern and a "glasses pattern" added to the faces. A sample observation with added backdoors is shown in Figure 5.1.

Note that the backdoors are inserted into the data via "FaceApp" [Fac19]. Due to the nature of this app (faces are analyzed using machine learning and patterns are fitted individually), the backdoors are not entirely identical. In contrast to Chapter 4, where the backdoors are completely identical, we decided to use these "non-identical but similar" patterns to illustrate potential robustness of the backdoors.

All datasets are uploaded to Zenodo and achievable under the DOI *10.5281/zenodo.3774167*, as well as the model files *10.5281/zenodo.3774170* of the results. The matching of each model file to the concrete experiment is listed in the appendix under Figure 8.2.

(a) Full beard backdoor (b) Glasses backdoor

Figure 5.1: Different backdoors added to the same observation

5.1 State of the art

In [GB12], the authors did an extensive comparison on classifying the Yale Face dataset using non-neural-network technologies. The best tested classification strategy was using the Fisherfaces method, which performs Principal Component Analysis as well as Linear Discriminant Analysis on the data. This method resulted in a testset accuracy of 95%. They resized the facial images to 56x46 pixels and used 5 observations of the dataset for training purposes.

For our experiments we rebuild a state of the art neural network used on this dataset first used by Manthouri et. al [KMT13]. They split each class containing 11 images into a training set of 9 observations and a testset of 2 observations. Their best results with this neural network achieved a testset accuracy of 80%.

We also split the dataset in the same way and started our experiments with centralized machine learning setting using for 100 training epochs. These results act as a reference value for further experiments with backdoors, and are displayed in Figure 5.2. As shown, we reach a testset accuracy of 92% which represents even better results than [KMT13]. A reason for this might be the augmentations to our training dataset in the prepossessing step described in Section 3.2.2.

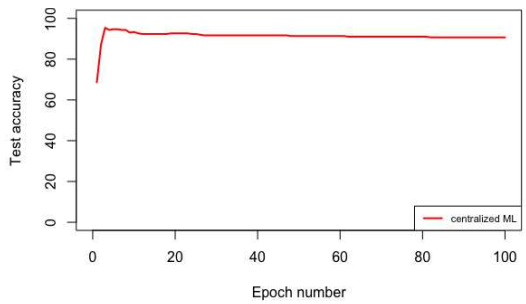
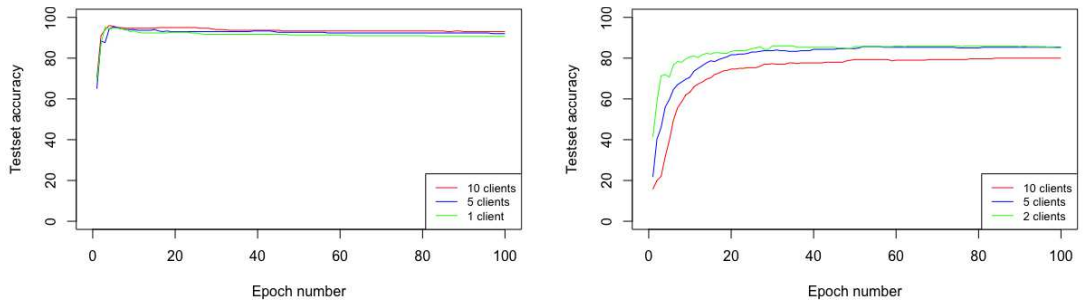


Figure 5.2: Results on the Yale Face dataset using centralized machine learning

In a federated learning environment, our baseline values to compare for the following backdoor attacks are a testset accuracy of 92% using sequential learning and 85% for federated averaging, each using 5 clients after 100 epochs showed, as depicted in Figure 5.3.



(a) Sequential learning

(b) Federated aggregation

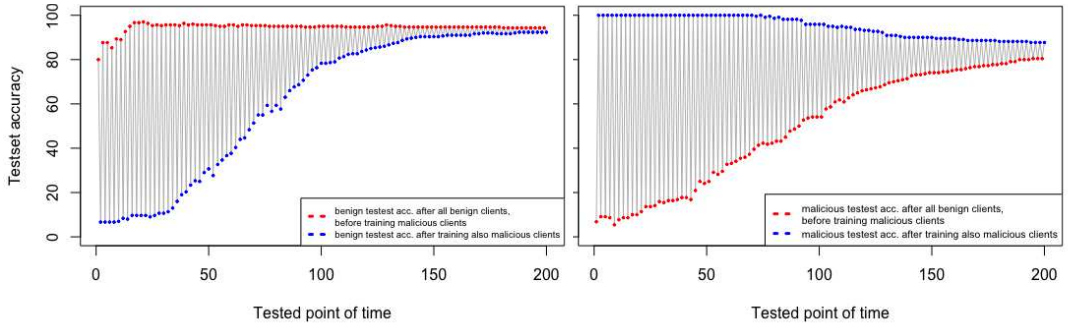
Figure 5.3: Our reference performance on different sized networks using sequential learning and federated averaging

5.2 Backdoor pattern: full beard

The first tested backdoor pattern is adding a full beard to two thirds of the classes. These classes are randomly selected and kept constant across all experiments. Classes selected for adding the backdoor are all except class 5, 7, 11 and 12. While persons of class 2, 9 and 13 are naturally wearing a small beard, class 7 wears a full beard similar to the one we added as backdoor. Each single observations of the poisoned classes is duplicated and equipped with backdoors, and we split the data following the principle as described in 5.1 into training and testset. We test the performance of the backdoor using networks with sequential and federated aggregation.

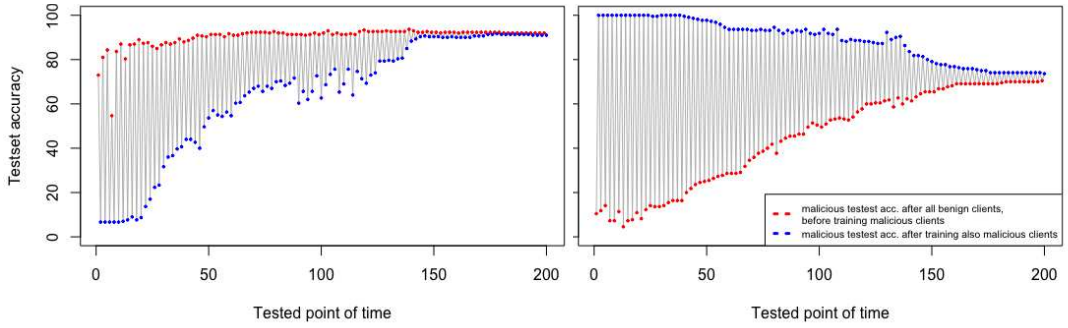
5.2.1 Sequential learning

We perform the experiments in the sequential learning setting on a network consisting of 4 benign (from now on referred as "smaller network") and 1 malicious clients, and a network consisting of 9 benign and 1 malicious clients (referred as "bigger network"). We also analyse the point of time the adversary is trained at.



(a) Benign testset (b) Malicious testset

Figure 5.4: Performance of one malicious client trained **after** 4 benign clients using full beard backdoor pattern on **sequential learning**



(a) Benign testset (b) Malicious testset

Figure 5.5: Performance of one malicious client trained **after** 9 benign clients using full beard backdoor pattern on **sequential learning**

Training adversaries after benign clients

In Figure 5.4 we observe the results of the smaller network having the malicious client trained last. Note that as described in more detail in the graphics of Section 4.4, the number of x-axis ticks equals 200 due to testing twice per epochs over 100 epochs. We can observe that in early stages of the model training, the performance on the benign and the malicious testset are fluctuating significantly. The reason for this behaviour might be the fact that the data itself differs very strong (having all available target classes in the benign clients vs. having only one class when the adversary is trained). With increasing time, the model apparently learns the backdoor and these fluctuations start to decrease. After all 100 training epochs, the performance on the benign testset has stabilized, and the performance on the malicious testset fluctuates only around 10% of the malicious testset accuracy. The accuracy on the benign testset sticks at 92%, while the accuracy on the backdoor task reaches 87%.

Comparing these results to the bigger tested network in Figure 5.5, we can clearly see

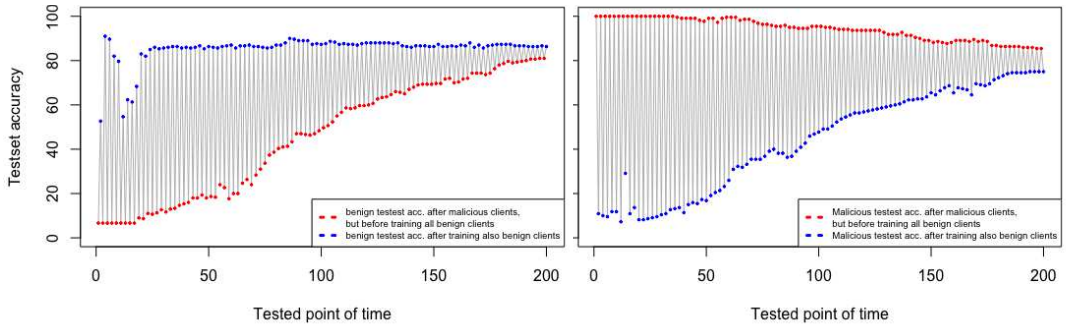
that in the latter case the fluctuations on the benign testset cancel our earlier. The fluctuations on the malicious testset are occurring at a comparable level as on the smaller network. In this testcase we can observe that the accuracy on the malicious testset has decreased to 72%. This accuracy drop might be due to the fact that in our experiment settings we assure that each client trains the model on the same amount of observations. The fraction of malicious clients training data is 1/5 in the small network but only 1/10 on the big network.

Training adversaries before benign clients

In Figures 5.6 and 5.7 the results of experiments are displayed where the adversary is trained before all benign clients. For the smaller network portrait in Figure 5.6 we observe fluctuations at the same level as in experiments when the adversary is trained last. Notably, the accuracy of the malicious testset is drastically reduced by around 20% comparing to Figure 5.4.

In the bigger sized network (Figure 5.7), the performance drop on the backdoor testset has increased comparing to the smaller sized network, leaving up a backdoor performance of only 57%. The accuracy on the main task dropped from 80% to 78%.

To conclude, we see that the bigger networks are generally performing worse in respect to backdoor accuracy, in comparison to the smaller networks. This might be due to the fact that, from a relative perspective, the fraction of malicious clients dropped from 20% to 10% of the clients.

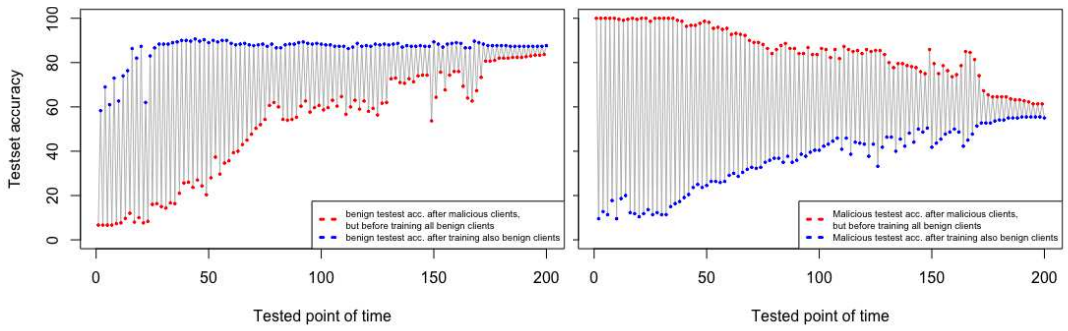


(a) Benign testset (b) Malicious testset

Figure 5.6: Performance of one malicious client trained **before** 4 benign clients using full beard backdoor pattern on **sequential learning**

Evaluating beard backdoor performance in sequential learning

In this section we evaluate the capability of the beard pattern to be used for backdoor attacks. In the Figure 5.8a the per class accuracies of each class are displayed in a bar-plot. The black bars represent our starting position - a network without any adversary. The



(a) Benign testset (b) Malicious testset

Figure 5.7: Performance of one malicious client trained **before** 9 benign clients using full beard backdoor pattern on **sequential learning**

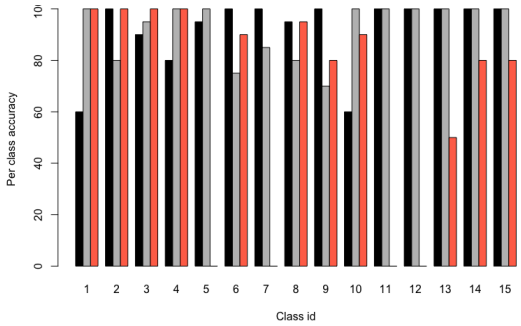
grey bar represents the per class accuracies in a sequential network consisting of 4 benign clients and 1 adversary which is trained at the end of the learning cycle.

While the accuracy has not changed or decreased for some classes, it is noteworthy that they even increased for class ids 1, 3, 4, 5 and 10. We believe that this, on the first glance, illogical behaviour is a result of some overfitting in the non-poisoned network. The non-adversarial network converges after only 10 epochs, and apparently the overall accuracy drops from 95% to 92% over the following 90 epochs (see Figure 5.3a). By adding malicious data this overfitting might be decreased (at least for some classes), as the network is busy with the poisoned data.

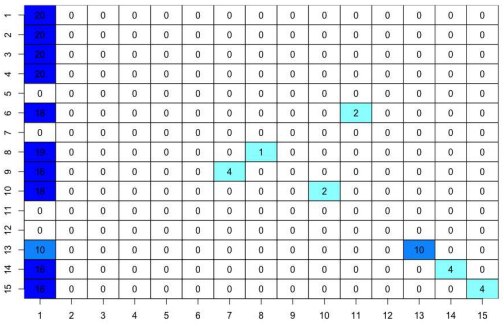
The red bars of this plot represent the accuracy of the poisoned data that are correctly misclassified as class 1. Classes 5, 7, 11 and 12 were not selected for adding a beard in the first place, and so they do not have backdoored data in the testset.

Overall seen, the backdoor is performing at an accuracy of 88%. Classes 9 and 13 are deceeding this value. Figure 5.8 represents the confusion matrix of the backdoored samples. If all backdoors were working at 100%, than each backdoored input sample (from the y-axis) would be matched to target class 1 in the x-axis. Most malicious samples are classified as expected or still as their originating class (meaning, the backdoor is not recognized). Notably, we also observe that four samples, originating from class 9, are misinterpreted to class 7. This is an interesting case because per default, the person of class 7 naturally carries a full-beard.

To conclude, we were able to enter the beard backdoor using sequential learning to a high level (in the best tested case at 81%) while also preserving huge parts of performance on the benign testset (77%). In general, the later an adversary attacks a learning cycle, the better the backdoor is implemented into to network, but the higher the performance is degraded on benign testdata. Opposing an adversary more benign clients, as well as using bigger networks, leads to less success in introducing the backdoor.



(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network



(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: predicted outcome for adversarial input data, y-axis: backdoor added to samples originating from corresponding classes - each labeled to target class id 1)

Figure 5.8: Comparing per class accuracies in a sequential network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - backdoor type: beard

5.2.2 Federated averaging

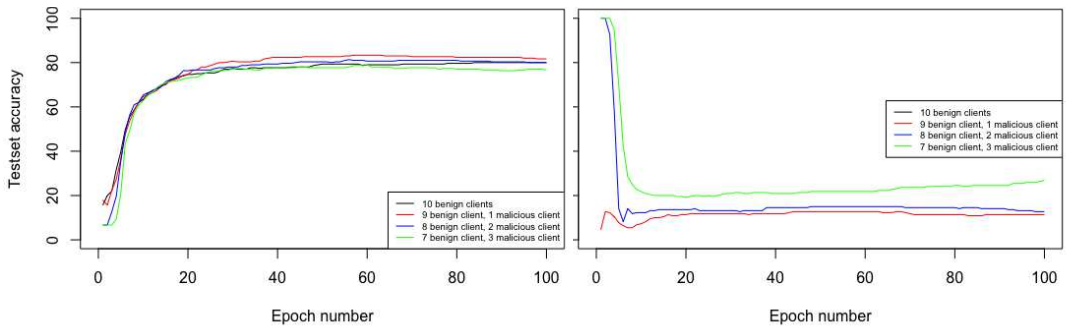
In this section we test the "beard-shaped" backdoors in federated learning environments. In contrast to sequential learning, we also experiment on varying fractions of benign to malicious data in the adversaries, as this is crucial when using model replacement strategy. In detail, if an adversary contains 50% benign and 50% malicious data, we create two pools - one containing all non-poisoned observations of all classes, and one containing all poisoned observations of all poisoned classes. Then, we randomly add observations to the client in a ratio of 1:1, until the client has as much observations as all other benign clients have.

Basic attack strategy

For the basic attack strategy, we test a network consisting of 10 clients at different numbers of adversaries. In all tested cases, the attackers contain 100% poisoned data. The results are depicted in Figure 5.9. As one can see, the performance on the benign testset stays the same throughout all tested cases, and does not deviate significantly from a network without adversaries. The accuracy on the malicious testset increases with the the more attackers participate in a network, but sticks in any tested case under 30%. At this low accuracy the attack can be considered not successful.

Model replacement strategy

In the model replacement strategy we first try a small network consisting of 4 benign and 1 malicious client (Figure 5.10). Here, we focus on testing fractions of benign to malicious data, as they are very relevant to the success.



(a) Benign testset (b) Malicious testset

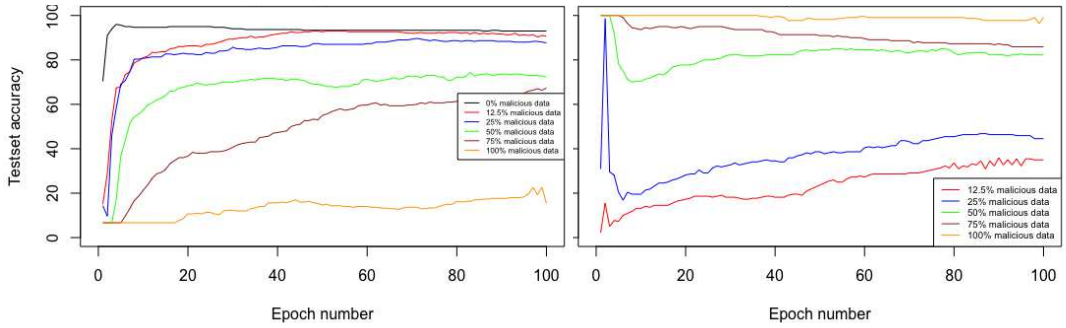
Figure 5.9: Performance of backdoors using **basic attack strategy** in a network of 10 clients using full beard pattern

We can see that test cases of 12.5% and 25%, the performance of the benign testdata is nearly identical to a network without adversaries after 100 epochs. The performance on the malicious testset however is around 35% for the smaller (12.5%) and 44% for the bigger (25%) tested fraction. For a fraction of 50% poisoned data, the performance on the benign testset lies at 72% while the accuracy on the malicious testset sticks around 82%.

Obviously, there is a clear trade-off for the fraction of malicious data in adversaries. The higher this fraction, the better the backdoor is implemented into the model, but the worse it performs on benign testdata. This trend still continues when the fraction is increased to 75% and 100% malicious data in malicious clients.

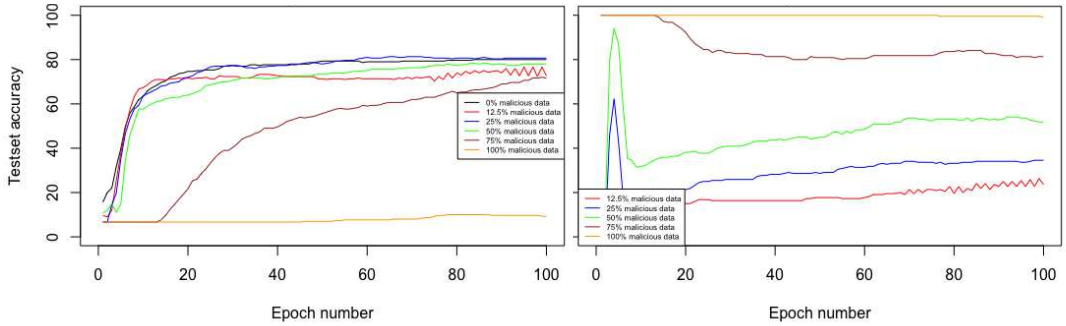
In a bigger network consisting of 9 benign and 1 malicious client (Figure 5.11), we observe the same pattern. The higher the fraction of malicious data in malicious clients is, the better the backdoor is inserted into the global model, but the worse the model works on non-poisoned data.

In general, the performance on the benign testdata is lower on the bigger sized network, in comparison to the smaller network. The model replacement attacks, in contrast, perform in both networks at comparable levels.



(a) Benign testset (b) Malicious testset

Figure 5.10: Performance of backdoors using **model replacement strategy** in a network of 5 clients (4 benign and 1 malicious) using full beard pattern



(a) Benign testset (b) Malicious testset

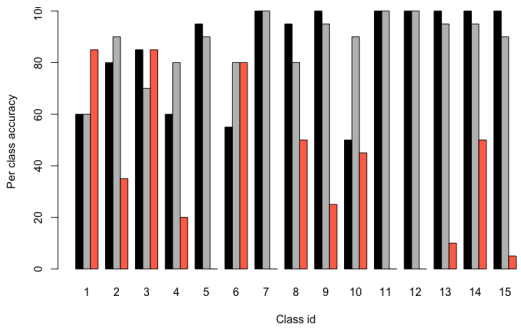
Figure 5.11: Performance of backdoors using **model replacement strategy** in a network of 10 clients (9 benign and 1 malicious) using full beard pattern

Evaluating beard backdoor performance in federated averaging

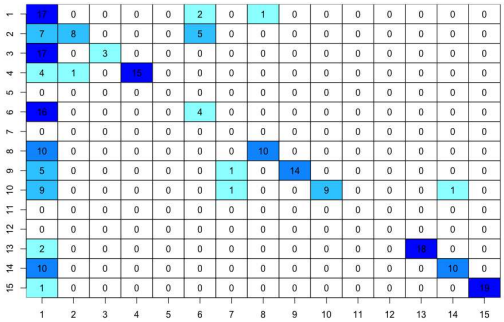
As done in the case of sequential learning, we investigate the per class accuracies of federated learning on benign and malicious data. To showcase, we use a network of 4 benign client and 1 adversary which uses the model replacement method. In Figure 5.12, the per class accuracies of an adversary using 25% malicious data are depicted. Overall, the backdoor is introduced at an accuracy of 45%, while benign testdata reaches an accuracy of 88%. We clearly see that most of the data is still classified as if the backdoors were not added, observable by the colored diagonal. This indicates that the backdoor is either not strong enough, the adversary uses a too small fraction of poisoned data or the model is not trained long enough for the backdoor to make it into the global model.

In Figure 5.13 we show the per class accuracies at an increased percentage of malicious data to 50% in the adversary. Now, most of the adversarial data (83%) is correctly misinterpreted as class 1. At the same time, the accuracy on the benign testset dropped from 85% to 72% - a drop that mainly results from benign test-samples also classified as class 1.

Resulting from these experiments, our federated learning setup is definitely capable of being attacked by adversaries when beard patterns are added. In the following section we investigate a "glasses" pattern and compare its effectiveness with the "beard" pattern to gain insights on the influence of the pattern's prominence.

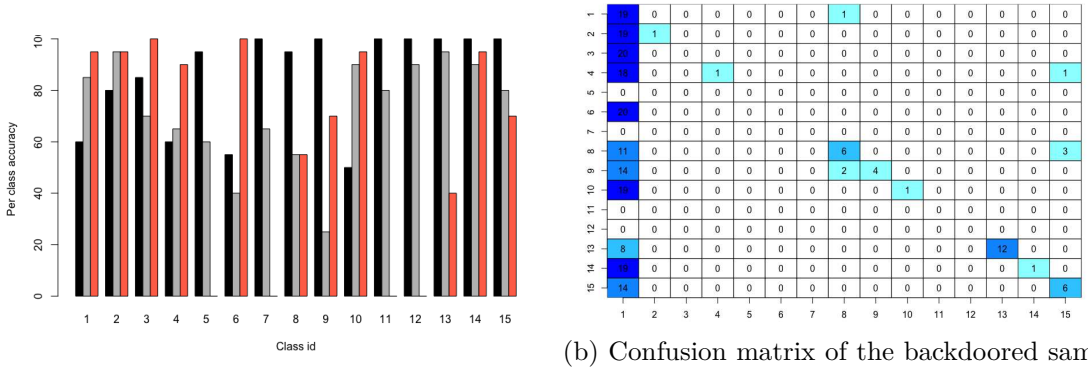


(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network



(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: backdoor added to samples originating from corresponding classes (each labeled to target class id 1), y-axis: predicted outcome for adversarial input data

Figure 5.12: Comparing per class accuracies in a federated averaging network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - fraction of malicious samples in adversary: 25%- backdoor type: beard



(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network

(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: backdoor added to samples originating from corresponding classes (each labeled to target class id 1), y-axis: predicted outcome for adversarial input data

Figure 5.13: Comparing per class accuracies in a federated averaging network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - fraction of malicious samples in adversary: 50%- backdoor type: beard

5.3 Backdoor pattern: glasses

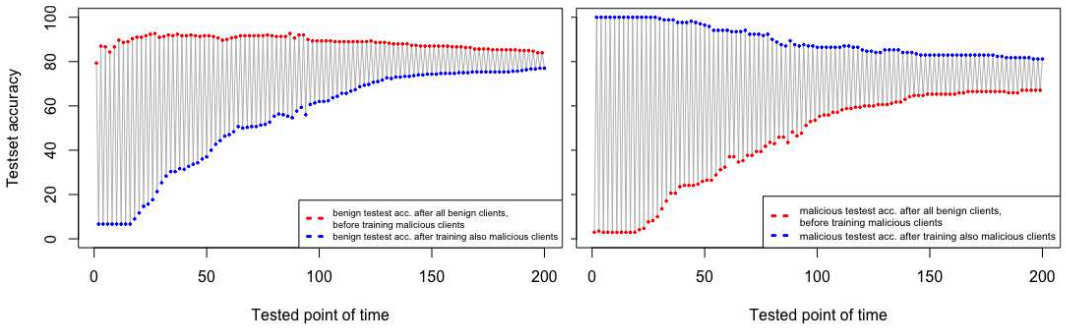
The second tested backdoor pattern are black glasses. Like the beard pattern, they are added to a random subset of two thirds of each person. Classes with added backdoors are the ones with the following ids: 2, 4, 6, 7, 8, 9, 10, 12, 14 and 15. Naturally, people wearing classes are represented by class id 8, 13 and (partly) 14. While glasses of class 13 and 14 are very thin, the glasses of class 8 resembles the big black glasses we use as backdoor. Experiments are again performed on a network consisting of 4 benign clients and 1 adversary (referenced as "small network"), and a network consisting of 9 benign clients and 1 adversary (respected as "bigger network"). The results of these experiments are discussed in the following sections.

5.3.1 Sequential learning

Training adversaries after benign clients

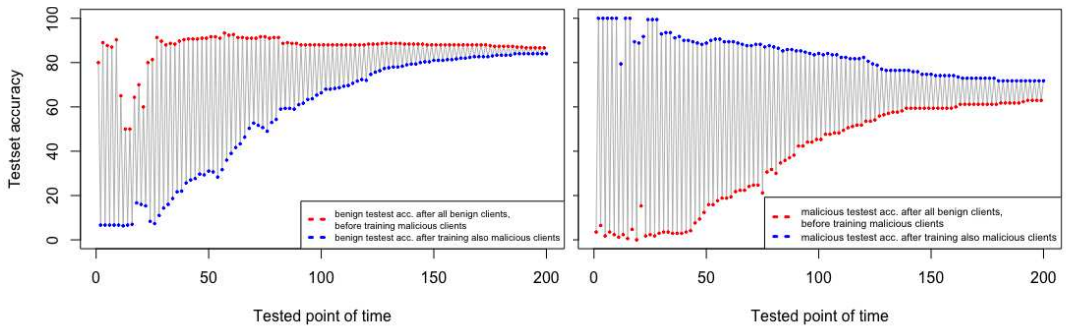
In Figure 5.14 we observe, comparable to the beard backdoor, big fluctuations in the benign as well as malicious testset, when the adversary retrains the model after all benign clients. These fluctuations start to become smaller the more training epochs have elapsed. After the final epoch, they still fluctuate in an accuracy of around 5% in the case of the benign testset, and around 10% on the malicious testset. In general, the final accuracy on the benign testset is 77%, while the backdoor's accuracy settles at 81%. When the adversary is trained after all benign clients in the bigger network, as seen in Figure 5.15, we also observe these fluctuations in accuracy. However, in the case of the benign testset, they start to settle earlier and are nearly stable after 100 training epochs. With a benign testset accuracy of around 84% the performance on the main task increased, while it decreased to 71% on the backdoor task.

In terms of benign testset accuracy, the bigger network outperforms the smaller network. In respect to malicious testset accuracy, it is the other way around.



(a) Benign testset (b) Malicious testset

Figure 5.14: Performance of one malicious client trained **after** 4 benign clients using glasses backdoor pattern on **sequential learning**



(a) Benign testset (b) Malicious testset

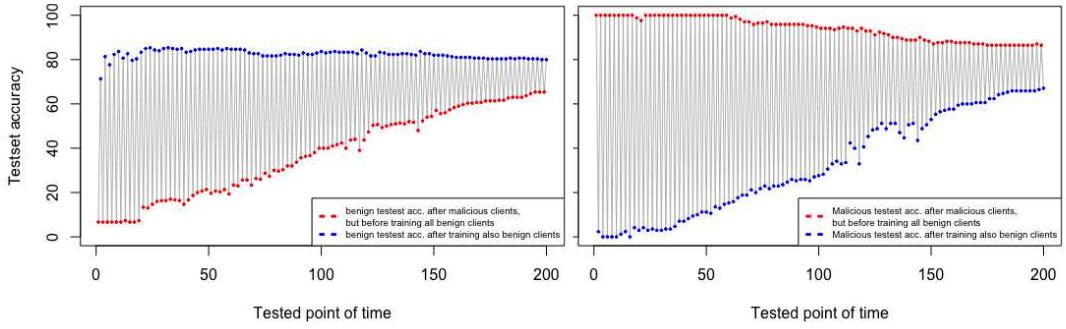
Figure 5.15: Performance of one malicious client trained **after** 9 benign clients using glasses backdoor pattern on **sequential learning**

Training adversaries before benign clients

In the smaller network, training adversaries first results in big accuracy fluctuations. Even after 100 epochs, the accuracy fluctuates around 15% on the benign task, and even 20% on the malicious task. After finishing the training cycle, they reach an accuracy of 80% on the main task and 67% on the backdoor task. This is displayed in Figure 5.16.

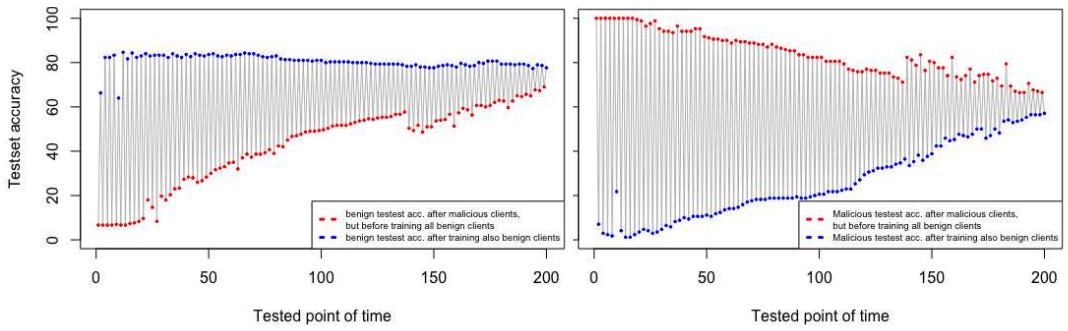
These fluctuations are still present in the bigger network (Figure 5.17). The accuracy on the main task settles around 78%, while the accuracy on the backdoor drops to 57%. This drop of performance on the malicious testset also occurs when using the beard backdoor pattern.

Observations on this backdoor pattern agree with observations achieved with the beard backdoor. Throughout all tested cases, the accuracy on the glasses backdoor is lower than using the beard pattern in the sequential learning setup.



(a) Benign testset (b) Malicious testset

Figure 5.16: Performance of one malicious client trained **before** 4 benign clients using glasses backdoor pattern on **sequential learning**



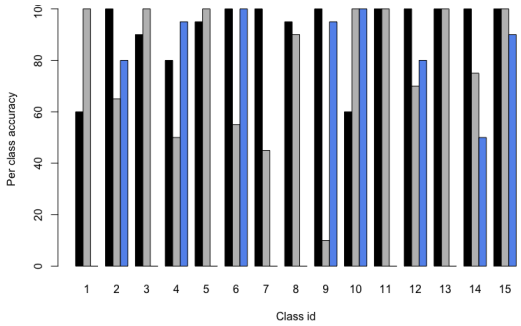
(a) Benign testset (b) Malicious testset

Figure 5.17: Performance of one malicious client trained **before** 9 benign clients using glasses backdoor pattern on **sequential learning**

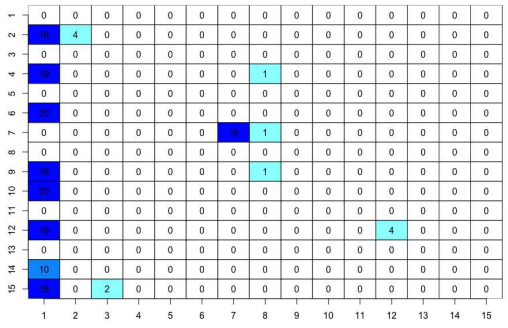
Evaluating glasses backdoor performance in sequential learning

By evaluating the backdoor performance, we compare the overall accuracy of benign testdata in a non-poisoned sequential learning network (92%) to the benign accuracy in a poisoned network (92%). Note that both networks consist of 5 clients, and one of these clients is an adversary in the case of the poisoned network. Overall, the accuracy on the benign testdata stays constant. By looking at the per class accuracies however in Figure 5.18a (black: non poisoned network, grey: poisoned network) we see than a part of them has changed. While one would expect that per-class accuracies stay at a constant level or decrease a bit, they are even increasing for some classes. This is also happening for the beard classes, and we believe that it is a result because of overfitting in the non-poisoned network.

The overall backdoor accuracy measures 88%. Obtained by analysing the blue bars in Figure 5.18a the biggest outlier is class 7. By looking at the confusion matrix in Figure 5.18 we observe that the backdoor is not working for this class, as the test data is still classified as if the backdoor is non existent. Also, even tough our backdoor is similar to the glasses naturally existing in class 8, there are only two single observations are predicted to be part of this class.



(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network



(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: predicted outcome for adversarial input data, y-axis: backdoor added to samples originating from corresponding classes - each labeled to target class id 1)

Figure 5.18: Comparing per class accuracies in a sequential network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - backdoor type: beard

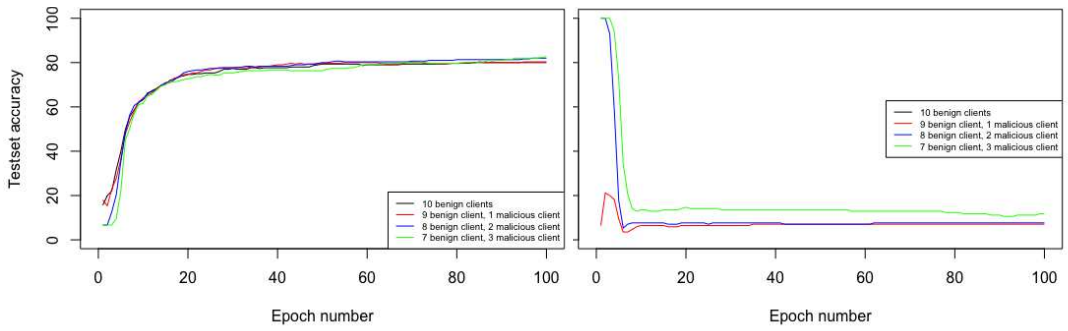
5.3.2 Federated averaging

In the setting of federated averaging, we again split our experiments into two sections, representing both tested attack strategies.

Basic attack strategy

For this strategy we set up a testing environment with a network consisting of 10 clients in total. Then, we replace one, two or three clients with adversaries. Each adversary trains the local model with 100% malicious data (while keeping the number of observations for training the same as benign clients). In Figure 5.19 we summarized the observations.

The performance on the benign testdata is nearly identically to a network without adversaries. The backdoor, on the other hand, is not introduced at all for any tested case. While there is a slight trend of accuracy increase with a higher number of attacker, the highest tested value of 3 adversaries (vs. 7 benign clients) reaches a malicious testset performance of around 15%. This low performance of the basic attack strategy is also observed using the beard-backdoor pattern. It seems that the glasses backdoor is even performing worse than the beard backdoor, which resulted in an accuracy of around 30% in the same scenario.



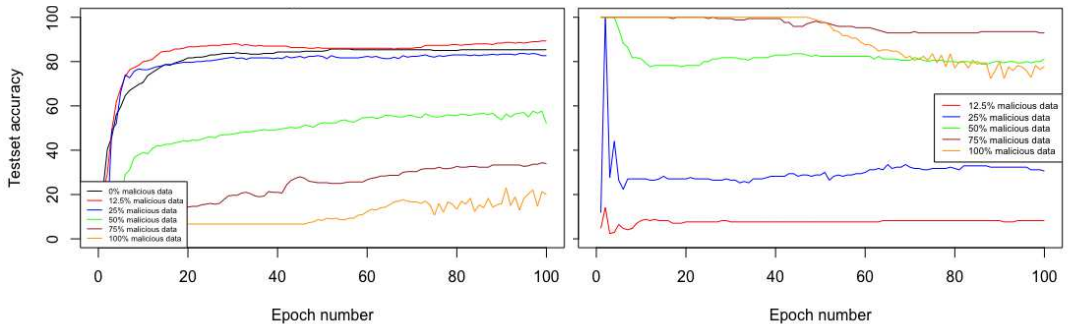
(a) Benign testset (b) Malicious testset

Figure 5.19: Performance of backdoors using **basic attack strategy** in a network of 10 clients using glasses pattern

Model replacement strategy

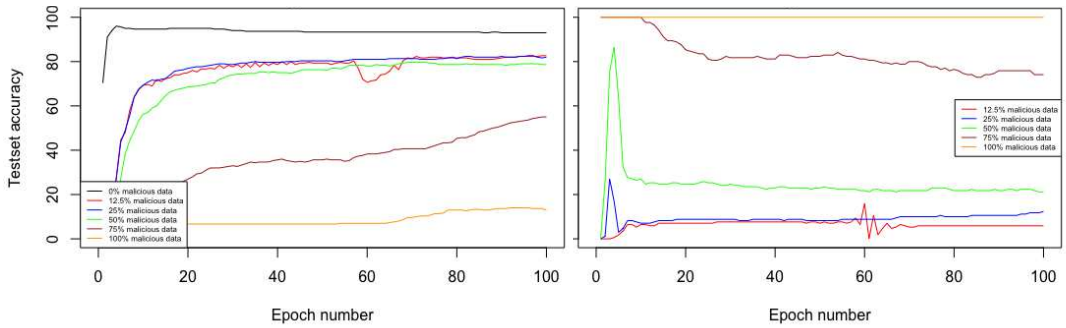
For the model replacement strategy, the results are represented in Figure 5.20 on a small network consisting of 4 benign and 1 malicious client. We again observe the same trade-off behaviour concerning the fraction of malicious data in adversaries. A higher fraction leads to a worse performance on the benign testset, which is connected to a better performance on the malicious data.

Figure 5.21 represents experiments on a bigger network consisting of 9 benign clients and 1 attacker. Experiencing the trade-off pattern again, in respect to the experiments on the "full beard" pattern the backdoor accuracy is generally much smaller. In other words, comparing the beard and the glasses patterns the same fraction of malicious data in adversaries, the accuracy on the malicious testset is consistently lower when using the glasses pattern.



(a) Benign testset (b) Malicious testset

Figure 5.20: Performance of backdoors using **model replacement strategy** in a network of 5 clients (4 benign and 1 malicious) using glasses pattern



(a) Benign testset (b) Malicious testset

Figure 5.21: Performance of backdoors using **model replacement strategy** in a network of 10 clients (9 benign and 1 malicious) using glasses pattern

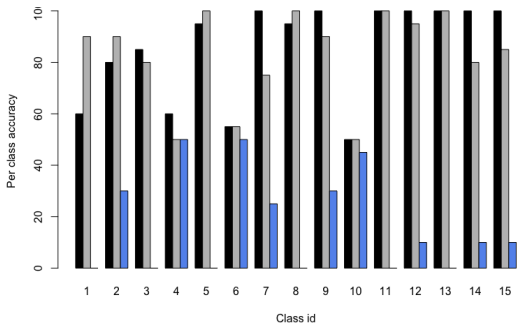
Evaluating glasses backdoor performance in federated averaging

In a federated averaging network consisting of 4 benign clients and 1 adversary at 25% malicious data, the overall accuracy on the benign testset is 82%. Compared to a non-poisoned scenario, this represents a performance drop of 3%. Comparing the per class accuracies, depicted in Figure 5.22a, the variations are also small. On some classes we again see a higher performance than in a non-poisoned network. The overall backdoor performance amounts 31% - a comparably low value. By looking at the confusion matrix in Figure 5.22b, most of the backdoored data is still classified as if there was no pattern added. Apparently, the global network has learned the backdoor to a too small degree.

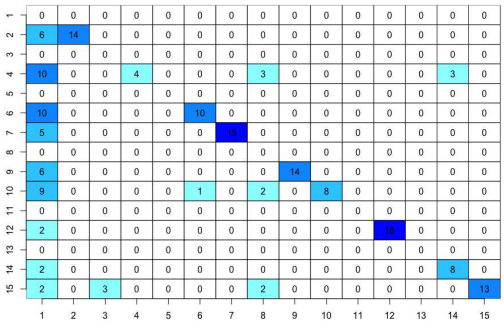
When the relation of malicious data to benign data is increased to 50% in the adversary, the backdoor performance increases to 81%. Now, most of the backdoored classes are classified correctly, as depicted in Figure 5.23b. Simultaneously, the overall accuracy of the benign testset dropped to 52%. By looking at the per-class accuracy graph in Figure 5.23a we observe that especially the classes with a high accuracy in the malicious testset result in a lowered performance on the benign testset.

Comparing the results using the glasses pattern to results obtained by the beard pattern, the performance drop using the glasses backdoor (81% to 52%) exceeds the drop using the beard backdoor (82% to 72%) - at comparable accuracies on the malicious test-sets.

Coming to a conclusion, we observe that we were able to introduce the glasses backdoor into out global model, although at a lower level of accuracy. Potentially, a reason for this is that the pattern in general constitutes a smaller area in contrast to the beard backdoor.

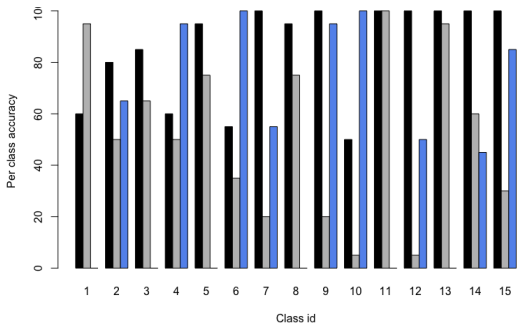


(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network

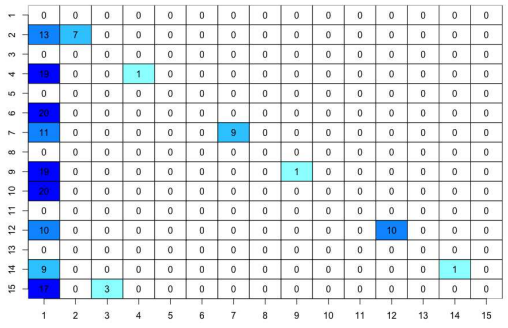


(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: backdoor added to samples originating from corresponding classes (each labeled to target class id 1), y-axis: predicted outcome for adversarial input data

Figure 5.22: Comparing per class accuracies in a federated averaging network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - fraction of malicious samples in adversary: 25%- backdoor type: glasses



(a) Per class accuracy on testset (black: benign samples in non-poisoned network, grey: benign samples in poisoned network, red: backdoored samples in poisoned network



(b) Confusion matrix of the backdoored samples in poisoned network (x-axis: backdoor added to samples originating from corresponding classes (each labeled to target class id 1), y-axis: predicted outcome for adversarial input data

Figure 5.23: Comparing per class accuracies in a federated averaging network consisting of 4 benign and 1 malicious client (adversary trained at the end of the learning cycle) - fraction of malicious samples in adversary: 50%- backdoor type: glasses

5.4 Attack analysis

In this case study we investigated a very practically relevant scenario: the classification of facial data. This is especially relevant in buildings or areas with access control via face recognition, which are on the rise with progressing digitisation. In contrast to prior work (e.g. [WYS+19]) we focus on adding backdoors that could be realistically feasible and are inconspicuous.

Overall seen, we conclude that using this state of the art neural network the introduction of both backdoors works to a high degree - with the full beard backdoor pattern performing a at higher level. As already mentioned, this is likely due to the bigger size of full beard pattern compared to the glasses.

Summing up perceptions on the research questions, we were again able to show that in the sequential learning setting the point of time the adversary trains his model matters to a high degree. Moreover, we were able to see that the model replacement strategy outperforms the basic attack strategy drastically, and that the success of former depends on the fraction of malicious to benign training data in the adversary. Generally, our observations match with the results of the case study on traffic signs in Chapter 4.

In the next Chapter we evaluate the results of both case study results with former observations of literature to finalize our research questions.

CHAPTER 6

Evaluation of results

In this chapter we investigate the research questions formulated in Section 1.3 based on existing observations in literature as well as our results.

6.1 Evaluation of number of clients

The first research question focuses on the effect of a varying number of clients in the machine learning process. In centralized machine learning, all data is processed on one single, client, while the two types of federated learning usually consist of multiple clients.

6.1.1 Observations by literature

Sheller et. al. [SRE+19] experiment on the performance of different techniques of federated learning on the classification of a brain tumor image dataset. Federated averaging and incremental sequential learning approaches are both tested on the following numbers of clients: 4, 8, 16 and 32. Also, they test a central machine learning setting on one single client processing all data.

They measure a model's performance using the Dice Coefficient, also known F1 Score. This effectiveness-measurement represents the harmonic mean of precision and recall, therefore a higher value means better performance. In Figure 6.1 their experiments are depicted. As represented in the legend, "FL" stands for "federated averaging", "CIIL" equals "incremental sequential learning" and "Data-Sharing" a centralized training model. The shading represents the min /max values for the tested method. Overall seen, federated learning techniques perform nearly identical to a centralized approach.

Split into graphs and grouped by different numbers of clients, the results in Figure 6.2 give some deeper insights.

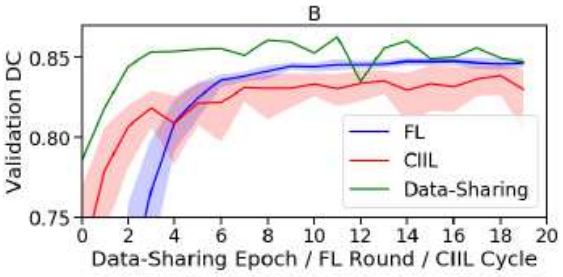


Figure 6.1: Comparison of experiment results on different learning approaches by Sheller et. al., graphic by [SRE+19]

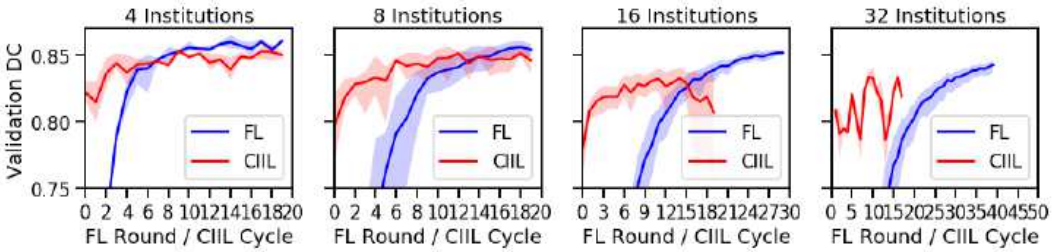


Figure 6.2: Performance of federated averaging and sequential learning approaches on experiments by Sheller et. al., graphic by [SRE+19]

Considering only the results of incremental sequential learning, it appears that at higher numbers of clients (tested values 16 and 32), the overall model performance is drastically fluctuating. They follow that a reason for this might be "catastrophic forgetting" [KPR+17]. This behaviour describes the phenomenon, that in later stages of the learning cycle the model has already "forgotten" what it learned in early stages. This effect increases with a large number of clients and small amounts of data per client. With a smaller amount of clients, the model performance is nearly identical (F1 Score: 0.843 at 4 clients, and 0.839 at 8) with the centralized model (0.862).

Federated averaging, however, suffers from far less instabilities. The overall performance is growing steadily, especially at a higher number of clients, and overall seen, it is outperforming sequential cyclic learning (F1 Score: 0.862 at 4 clients, 0.865 at 8). In fact, it achieves nearly identical performance compared of the model performance of the centralized model. On the down side however, it takes longer to converge, especially at an increasing number of clients.

6.1.2 Evaluation of our results

Comparing the sequential network to the one with federated aggregation, we also observe in our experiments that the federated aggregation network takes more time to converge.

What we do not observe is that the level of effectiveness in federated aggregation is higher than in sequential learning, because in most cases they are performing identically.

In a sequential learning setting, our baseline experiments on the number of clients are performed on iid data, so we do not observe the forgetting phenomenon in this case. We investigate data distributed in a non-iid way in Section 6.2 in more detail will find out that our experimental results match with the behaviour shown by Sheller et. al. [SRE+19].

Addressing a network where the data is combined using federated aggregation, we can observe the same behaviour as Sheller et. al. [SRE+19]. The more clients are participating in a federated aggregation network, the more training epochs the global model needs to converge.

6.2 Evaluation of distribution of data

Due to the nature of federated learning, it is likely that data is distributed in a non i.i.d. manner.

6.2.1 Observations by literature

McMahan et. al. [MMR+17] experiment the MNIST dataset [LC10], a dataset on handwriting classification, using the federated averaging algorithm. Their results show that non-iid data needs more time to reach a certain accuracy threshold.

Shoham et. al. [SAK+19] also work on training the MNIST dataset on non-iid data, in the setting of federated averaging as well as sequential learning. They result, in both cases, in a reduced performance on highly skewed non-iid data, in terms of effectiveness and the number of communication rounds needed, compared to iid data.

For sequential leaning, their declared reason for decreased performance is catastrophic forgetting, the phenomenon of losing knowledge on (long) ago trained data, as described in the previous section.

In federated aggregation, they draw connections to "Lifelong Learning", which is also deeply related to catastrophic forgetting. It describes the challenge of learning "Task A", and afterwards learning "Task B" but without forgetting what has already been learned. While in federated aggregation this sequential learning is replaced by learning parallel, they state that both learning types face the same problem, namely "how to learn a task without disturbing different ones learnt on the same model".

The authors even propose a possible solution to overcome these shortcomings in form of adding "a penalty term to the loss function to compel all local models to converge to a shared optimum."

6.2.2 Evaluation of our results

We can confirm the results of [SAK+19] regarding sequential learning as we also observe in both case studies that it heavily depends on the point of time the “exclusive” classes are trained

Adressing federated aggregation, we can also confirm observations by Shoham et. al. and McMahan et. al., as our results also lead to a reduced performance on non-iid data in comparison to iid data.

6.3 Evaluation of timing of the attack (in sequential learning)

This research question is situated in a similar area like the research question in Section 6.2. The timing of a backdoor attack in a sequential learning setting describes the point of time an attacker attends in the "learning cycle" and enters its poisoned data.

6.3.1 Observations by literature

To the best of our knowledge, there is no paper that deals with backdoor attacks in a sequential learning are yet. In the following, we describe some papers that deal with backdoor attacks in a centralized ML setting, which are comparable to sequential learning to some extend.

Earlier work like [GDG17], [WYS+19] use the following approach: First, they poison a subset of the training data set (ranging from 10% to 50%) and train this modified dataset for many consecutive epochs. Using this strategy, they were able to implement backdoors into the model, without significantly decreasing the accuracy on the main task. The training process is centralized, and the poisonous data is processed during the whole training process.

Shoham et. al [SAK+19] show catastrophic forgetting in sequential learning in general, without handling backdoor attacks.

Catastrophic forgetting is not a phenomenon of sequential learning only, as it is also happening in centralized learning. According to Kemker et. al. [KMA+18], catastrophic forgetting is described in centralized learning. In general, neural networks need to be "malleable" to learn new tasks on the one hand - but bigger weight updates cause losing previous knowledge on the other hand. Weights that are "too stable" prevent the model of acquiring new tasks.

6.3.2 Evaluation of our results

In our experiments we observe that whatever (benign or backdoored data) is trained at later stages of the training cycle, the more impact this data has on the "final model" at the end of the learning cycle. In our concrete test-settings however, we were still able to implement backdoor patterns even if they were trained in the beginning.

Note that the phenomenon of catastrophic forgetting occurs to be increasing the smaller the batches of non-iid data, and therefore the more often the optimizer is executed consecutively with the similar data.

6.4 Evaluation of the size and shape of the backdoor

In this section the prominence of the backdoor patterns in is evaluated. Both, poisoning traffic signs as well as images of faces has already been done with different backdoors, which will be described explicitly and finally compared to our results.

6.4.1 Observations by literature

In Section 2.6.2 and Section 2.6.3 several existing attacks are described in detail. In the following we will focus on backdoor patterns used on datasets for classification tasks on traffic signs and face images.

Gu et al. [GDG17] perform backdoor attacks on an American traffic sign dataset. They experiment with three different backdoor patterns, as depicted in Figure 6.3. Their sizes are "roughly the size of a Post-it note". We enumerate this vague size-description to between 1 and 2 percent of the traffic sign's area by analyzing their images, as they have not explicitly stated. All of their tested backdoors succeed in introducing the backdoor into the global model in an interval of 90% to 95% malicious testset accuracy.

Figure 6.3: Backdoor patterns used in experiments by Gu et al., graphic also by [GDG17]

Other observations by Wang et. al. [WYS+19] also deal with backdoor attacks on image classification tasks. They train, in a centralized environment, a model from scratch with both malicious and poisoned data. One of their tested datasets is the German Traffic Sign Recognition Dataset (which is also used in our experiments). They use a white squared backdoor with a relative size of 1% of the image which is statically located in the bottom right corner of the image. (Note: In our opinion this seems not to be a realistic assumption as the backdoor is constantly out of the actual traffic sign.) They are depicted in Figure 6.4.

Detailed configuration:

- centralized learning
- neural network architecture: custom (6 Conv + 2 Dense layers)
- trainind epochs: 10
- batch size: 32

- optimizer: Adam
- learning rate: 0.001
- image size: 32x32
- poisoned data: 10%

Also, they add the same backdoor on face classification datasets, again in bottom right corner. We do not consider to use squared-sized patterns in our experiments with the Yale face dataset as this scenario does not seem to be realistic, and it is noticeable by the user and suspicious.

(b) GTSRB (c) YouTube Face (d) PubFig

Figure 6.4: Backdoor patterns used in experiments by Wang et al., graphic also by [GDG17]

6.4.2 Evaluation of our results

We cannot totally compare the traffic sign results of Wang et. al. [WYS+19] to our results, but we can derive some relations. We can relate their centralized machine learning setting to some extend to our results obtained in the sequential learning setting, as there are similarities. In their approach they also use small batches, all trained each epoch - the same as we do in our sequential learning settings. (Note: we do group all malicious batches either in the beginning or the end of each epoch, but after 100 epoch this difference is negligible. Compare figures 4.34 and 4.35.)

In our setting we use 1% sized green patterns and additionally vary their positions. Our results differ only to some percentage points from theirs. We are obtaining an accuracy of around 95% on the benign testset and nearly 100% on malicious data, as opposed to the benign testset accuracy of 96% and malicious testset accuracy achieved by Wang et. al.

Apart from that, there is, to the best of our knowledge, no literature specifically addressing the prominence of the backdoors.

What we observe is an impact of the backdoor's size and rarity in the main task. In general, bigger patterns, as well as patterns with greater rarity (in terms of color) in the benign training dataset lead to an increased performance on the backdoor task.

6.5 Evaluation of different attack strategies

This section deals with the attack strategies described in Section 2.6.1. For evaluation we refer to the literature observations described in Section 2.6.3.

6.5.1 Observations by literature

Bagdasaryan et. al. [BVH+18] were testing both strategies on the "CIFAR-10" dataset [KNH12]. They distribute data over a set of clients consisting of p% malicious and (1-p)% benign clients. Each round, a subset of all clients is randomly selected and used for the specific epoch. Afterwards, they are trained concurrently and merged using federated aggregation.

They conclude that the model replacement method clearly outperforms the basic attack strategy, as depicted in Figure 6.5. For the replacement strategy, they need only 10% attackers to achieve a backdoor accuracy of around 95% on all tested backdoors. Using the basic attack, even the second highest tested value of 50% attackers only resulted in a backdoor accuracy of around 80%, depending on the used backdoor.

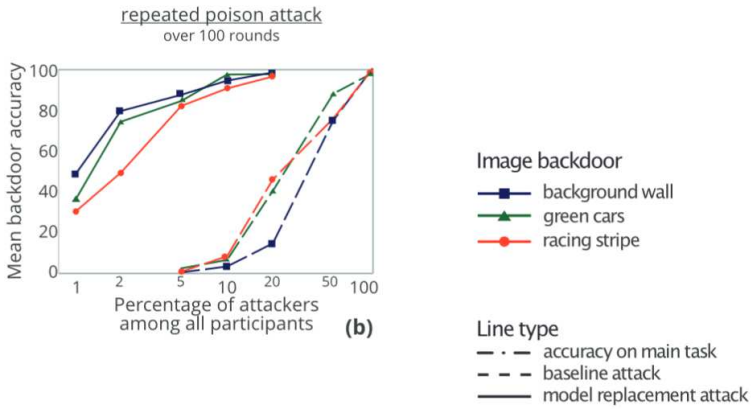


Figure 6.5: Comparison of basic attack strategy and model replacement strategy by Bagdasaryan et. al., graphic also by [BVH+18]

Nguyen et. al. [NRMS20] perform their experiments in a federated aggregation. Malicious clients are using the basic attack strategy, and they introduced the backdoor (at 100% accuracy) at a fraction of 15% poisoned clients.

6.5.2 Evaluation of our results

We can confirm the observations of [BVH+18] and [NRMS20].

In our tested scenario, the success of the basic attack strategy also depends on absolute number of clients in a network, as well as the number of malicious clients in respect to benign clients. The higher the fraction of malicious clients is, the more we are able to introduce the backdoor. In general however, this strategy is performing poorly in comparison to the model replacement strategy, and worse than in Nguyen et. al. [NRMS20].

The mentioned model replacement strategy performs, overall seen, very well as we were able to introduce the backdoor into the global model to a high degree in each test scenario. Note that a relevant criterion for the success is the fraction of malicious to benign samples in the adversaries. This fraction acts as a kind of trade-off between the performances on the benign and malicious testset. The more backdoored data an adversary contains, the better the effectiveness on the malicious testdata is, but the lower it is on the benign testdata.

6.6 Evaluation of number of attackers in relation to benign clients

A key question to the success of backdoor attacks is probably the fraction of attackers in a federated machine learning network. We evaluate attacks from literature in Section 2.6.2 and Section 2.6.3 regarding this aspect.

6.6.1 Observations by literature

Gu et al. [GDG17] experiment in a centralized ML environment. Instead of varying the fraction of attackers, they vary the fraction of poisoned data. They conclude that a higher fraction of backdoor images increases performance on the backdoor task while decreasing performance on the main task. A graph supporting their observations is depicted in Figure 6.6.

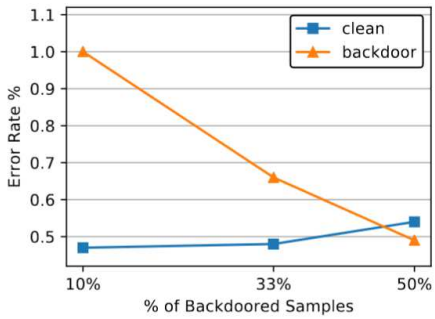


Figure 6.6: Impact of the proportion of backdoored samples in the training set on the error rate for clean and backdoored images by Gu et al., graphic also by [GDG17]

Wang et. al. [WYS+19], also using a centralized machine learning setting, keep their malicious data fraction constant at a value of 10%.

Bagdasaryan et. al. [BVH+18] test in a federated aggregation network and test the following percentages for the number of malicious clients in relation to benign clients: 1%, 2%, 5%, 10%, 20%, 50% and 100%. Throughout all their experiments, they keep the number of poisoned samples in the adversaries fixed (at 20 out of 64 observations per batch). Their results depend on the used attack strategy, as already discussed in Section 6.5. It can generally be said that a higher number of clients leads to a better performance of the backdoor task. The accuracy on the benign data is not negatively affected with an increasing number of clients.

Experiments by Sun et. al. [SKSM19] in a federated aggregation network using model replacement strategy conclude that their backdoors need at least 1% malicious clients (containing 100% poisoned data) be considered as successful in their scenario. Also, the higher the number of malicious clients is, the better the attack is performing.

Nguyen et. al. [NRMS20] are testing on federated aggregation and perform the most comprehensive evaluations of different numbers of clients, as well as different number poisoned data in the malicious clients. They are using a basic attack strategy, and their results are depicted in Figure 6.7.

Their observations show that one must distinguish between the number of attackers (which are tested at varying amounts of poisoned data), and the number of overall poisoned samples introduced in the network (achieved by counting poisoned data across all 100 clients). As a meta-analysis, at a PMR of 15% they need 80% of the data in these malicious clients to be poisoned, to reach a backdoor accuracy of over 99%. This leads to 12% of poisoned data overall existent in the network. At a PMR of 30% they require a PDR of 20%, leading to a rate of 6% poisoned data in the whole network. At a PMR of 35% they only require a PDR of 10%, resulting in 3.5% poisoned data in the network. A clear trend, resulting that the more clients are malicious, the less poisoned data must be present in those clients to reach a backdoor accuracy of over 99%. Note: in all of their tested cases, the accuracy on the benign testdata was kept constant at 100% as it is in a non-poisoned network.

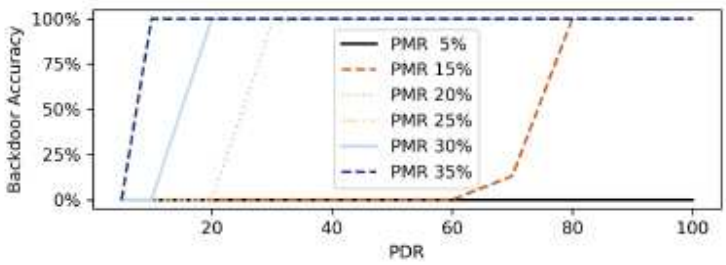


Figure 6.7: Backdoor accuracy for different poisoned data rates (PDR = percentage of malicious samples in malicious clients) and poisoned model rate (PMR = percentage of malicious clients in relation to benign clients). Note that the main task accuracy is 100% for all attacks. Graphic and data by Nguyen et. al. [NRMS20]

6.6.2 Evaluation of our results

In a federated learning network using sequential training we observed a high performance on the benign as well as malicious testset for all tested values of malicious clients in relation to benign clients. On the highest tested value of 20% we observe a slightly earlier convergence on the malicious testdata but a slightly later convergence on the benign testdata compared to a lower fraction of malicious clients. In general, it seems that the number of clients on the success of backdoor attacks in sequential learning plays only a comparably small role.

In an environment with federated aggregation we found that the basic attack strategy is less effective in introducing the backdoor into the global model than using the model replacement strategy. At the same number of clients, the accuracy on the malicious

testsets is significantly lower for both case studies. In comparison to Nguyen et. al. [NRMS20] we were not able to introduce a backdoor using the basic attack strategy - which might also be caused due to a very different setting of the machine learning task.

An important factor for the success of the model replacement method is the fraction between poisoned and non-poisoned samples in the adversary's training set. An increased fraction of poisoned data in comparison to non-poisoned data leads to a higher accuracy on the malicious testset. Simultaneously, the effectiveness on the benign testset drops significantly. A good value in our experimental setting amounts mostly between 25% and 50% poisoned data.

In the next chapter we summarize our work and provide an outlook into possibilities to extend this work and potential research topics for further investigations.

CHAPTER 7

Conclusions and future work

In this thesis we investigated the setting of federated learning. Its distributed nature enables many advantages. Among them is the possibility to directly utilize the clients' endpoints for training the machine learning models, instead of sending and processing all data on a centralized server, to potentially reduce costs. Furthermore, federated learning enables a higher level of privacy as sensitive data never leaves the client.

We examine two types of federated learning: sequential (incremental cyclic) learning, where a model is passed from one client to another for retraining purposes, and federation using an aggregator. In the latter strategy, the clients models are sent to an aggregation server where they are combined (e.g. by parameter averaging) into a global model.

A central topic of this thesis was investigating the influence of a varying number of clients in federated learning settings on the resulting models. We are able to confirm that in a sequential learning setting, independently from the number of participants, a machine learning model is able to be trained at the same level of effectiveness as in centralized learning. Also the second type of federated learning strategy, federated averaging, enables training a model at the same level of effectiveness. Notably, with a higher number of participants being averaged at the same time the convergence speed slows down significantly. In both cases, efficiency is very likely to decrease because of communication effects.

When investigating the ability of training non independent and identically distributed data, both federated learning techniques show weaknesses, as sparsely known classes are learned by a lower level.

Beside mentioned advantages, these federated learning techniques open up new possibilities for attackers, especially due to their distributed nature. We identified several attacks and focused on exploring backdoor attacks, a special form of model poisoning, more deeply. By entering a backdoor pattern into the data during the model training process, an adversary aims to enable targeted misclassification for all data poisoned with

this pattern during the deployment phase of the model. Non-altered data, however, shall still be classified correctly. Although the structure of the pattern is mostly clearly visible, its primary objective is to be as inconspicuous as possible (in respect to the data itself), while still being effective.

Two case studies investigating the possibility of backdoor attacks in image-data were performed. The first study addresses the classification of traffic signs, a typical task in machine learning. We processed the German Traffic Sign Benchmarks dataset [SSSI12] and added backdoors in different sizes and colors to the observations. To strengthen our observations we conducted experiments in the domain of facial recognition. For a realistic setting we chose backdoors in form certain glasses and a full beard.

Using observations on our studies as well as evaluating them against existing work we demonstrated that federated learning is highly susceptible for backdoor attacks. Investigations lead that the bigger and, in contrast to the data more "uncommon" a pattern is, the more effective the attack is. The number of attackers, the amount of poisoned data and, in the case of sequential learning the point of time the adversaries inject their backdoor are also key factors for the attacker's success.

Our work motivates to conduct further research in this field as we showed that federated learning can be seriously threatened by applying backdoor attacks. Future work should especially emphasize defense mechanisms. In a centralized setting there has been done some progress in averting model poisoning attacks, but new strategies have to be developed in order to deal with them in a distributed setting and its intrinsic assumptions.

CHAPTER 8

Appendix

8.1 Neural network structures

Listing 8.1: The architecture of the neural network used in the traffic sign case study

```
Net(
  (conv0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (bn0): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (conv1): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (pool_0): MaxPool2d(kernel_size=2, stride=2, padding=0,
     ↪ dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (conv3): Conv2d(96, 256, kernel_size=(3, 3), stride=(1, 1))
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (pool_1): MaxPool2d(kernel_size=2, stride=2, padding=0,
     ↪ dilation=1, ceil_mode=False)
  (dropout0): Dropout2d(p=0.37, inplace=False)
  (fc0): Linear(in_features=4096, out_features=2048, bias=True)
  (dropout1): Dropout2d(p=0.37, inplace=False)
  (fc1): Linear(in_features=2048, out_features=1024, bias=True)
  (dropout2): Dropout2d(p=0.37, inplace=False)
  (fc2): Linear(in_features=1024, out_features=43, bias=True)
)
```

Listing 8.2: The architecture of the neural network used in the face recognition case study

```
Net(
  (conv0): Conv2d(1, 6, kernel_size=(7, 7), stride=(1, 1))
  (bn0): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (conv1): Conv2d(6, 16, kernel_size=(8, 8), stride=(1, 1))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
     ↪ track_running_stats=True)
  (fc0): Linear(in_features=41616, out_features=15, bias=True)
)
```

8.2 Transformations for the yale face dataset

Listing 8.3: Dataset preprocessing used to enlarge the Yale Face dataset

```
#original
data_transform = transforms.Compose([
      transforms.ToTensor(),
      transforms.Normalize(
         mean=[0.485, 0.456, 0.406],
         std=[0.229, 0.224, 0.225]
      )])

#horizontal
data_transform = transforms.Compose([
      transforms.RandomHorizontalFlip(p=1.0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#brighter
data_transform = transforms.Compose([
      transforms.ColorJitter(brightness=(1.3,1.3), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#darker
data_transform = transforms.Compose([
      transforms.ColorJitter(brightness=(0.7,0.7), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])
```

```
#darker_horizontal
data_transform = transforms.Compose([
      transforms.RandomHorizontalFlip(p=1.0),
      transforms.ColorJitter(brightness=(0.7,0.7), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#brighter_horizontal
data_transform = transforms.Compose([
      transforms.RandomHorizontalFlip(p=1.0),
      transforms.ColorJitter(brightness=(1.3,1.3), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#veryBright
data_transform = transforms.Compose([
      transforms.ColorJitter(brightness=(1.6,1.6), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#veryDark
data_transform = transforms.Compose([
      transforms.ColorJitter(brightness=(0.4,0.4), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#veryBright_horizontal
data_transform = transforms.Compose([
      transforms.RandomHorizontalFlip(p=1.0),
      transforms.ColorJitter(brightness=(1.6,1.6), contrast=0,
         ↪ saturation=0, hue=0),
      transforms.ToTensor(),
      transforms.Normalize(...)])

#veryDark_horizontal
data_transform = transforms.Compose([
   transforms.RandomHorizontalFlip(p=1.0),
```

```
transforms.ColorJitter(brightness=(0.4,0.4), contrast=0,
    ↪ saturation=0, hue=0),
transforms.ToTensor(),
transforms.Normalize(...)])
```

8.3 Experiment result files

	A	B	C	D	E	F	G	H	I
1	Merge strategy	#benign clients	#malicious clients	distribution	% poisoned data	attack model	order of time	backdoor type	result file
2									
3	numbers of clients:								
4	sequential	1	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200225-091236
5	sequential	5	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200224-164321
6	sequential	10	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200224-112615
7	aggregation	2	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200225-185926
8	aggregation	5	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200225-152647
9	aggregation	10	0	equal	NULL	NULL	NULL	NULL	exp_gtsrb_20200302-140305
10									
11	Data distribution:								
12	sequential	10	0	80_20	NULL	NULL	minority first	NULL	exp_gtsrb_20200311-090415
13	sequential	10	0	80_20	NULL	NULL	minority last	NULL	exp_gtsrb_20200312-163037
14	sequential	10	0	50_50	NULL	NULL	minority first	NULL	exp_gtsrb_20200312-095036
15	sequential	10	0	50_50	NULL	NULL	minority last	NULL	exp_gtsrb_20200312-162036
16	sequential	20	0	80_20	NULL	NULL	minority first	NULL	exp_gtsrb_20200312-235421
17	sequential	20	0	50_50	NULL	NULL	minority first	NULL	exp_gtsrb_20200312-235504
18	aggregation	5	0	80_20	NULL	NULL	NULL	NULL	exp_gtsrb_20200318-110957
19	aggregation	5	0	50_50	NULL	NULL	NULL	NULL	exp_gtsrb_20200318-112229
20									
21	% of poisoned data								
22	sequential	4	1	equal	25	basic	first	green_1	exp_gtsrb_20200310-074846
23	sequential	4	1	equal	50	basic	first	green_1	exp_gtsrb_20200309-073105
24	sequential	4	1	equal	75	basic	first	green_1	exp_gtsrb_20200309-072909
25	sequential	4	1	equal	100	basic	first	green_1	exp_gtsrb_20200228-152630
26									
27	Order of time								
28	sequential	4	1	equal	100	basic	first	green_1	exp_gtsrb_20200228-152630
29	sequential	4	1	equal	100	basic	last	green_1	exp_gtsrb_20200229-071144
30	sequential	9	1	equal	100	basic	first	green_1	exp_gtsrb_20200308-160728
31	sequential	9	1	equal	100	basic	last	green_1	exp_gtsrb_20200305-234618
32	sequential	19	1	equal	100	basic	first	green_1	exp_gtsrb_20200308-161012
33	sequential	19	1	equal	100	basic	last	green_1	exp_gtsrb_20200310-152626
34									
35	basic attack:								
36	aggregation	4	1	equal	25	basic	NULL	green_1	exp_gtsrb_20200305-232824
37	aggregation	4	1	equal	50	basic	NULL	green_1	exp_gtsrb_20200305-055529
38	aggregation	4	1	equal	75	basic	NULL	green_1	exp_gtsrb_20200305-060337
39	aggregation	4	1	equal	100	basic	NULL	green_1	exp_gtsrb_20200318-203612
40	aggregation	9	1	equal	100	basic	NULL	green_1	exp_gtsrb_20200318-203814
41	aggregation	8	2	equal	100	basic	NULL	green_1	exp_gtsrb_20200319-161053
42	aggregation	7	3	equal	100	basic	NULL	green_1	exp_gtsrb_20200321-154836
43	aggregation	3	2	equal	100	basic	NULL	green_1	exp_gtsrb_20200320-085236
44	aggregation	2	3	equal	100	basic	NULL	green_1	exp_gtsrb_20200320-003449
45									
46	model replacement:								
47	aggregation	4	1	equal	12,5	replacement	NULL	green_0_5	exp_gtsrb_20200308-082156
48	aggregation	4	1	equal	25	replacement	NULL	green_0_5	exp_gtsrb_20200308-081922
49	aggregation	4	1	equal	50	replacement	NULL	green_0_5	exp_gtsrb_20200227-203454
50	aggregation	4	1	equal	75	replacement	NULL	green_0_5	exp_gtsrb_20200307-183049
51	aggregation	4	1	equal	100	replacement	NULL	green_0_5	exp_gtsrb_20200307-183021
52	aggregation	4	1	equal	12,5	replacement	NULL	green_1	exp_gtsrb_20200304-062118
53	aggregation	4	1	equal	25	replacement	NULL	green_1	exp_gtsrb_20200303-145824
54	aggregation	4	1	equal	50	replacement	NULL	green_1	exp_gtsrb_20200227-145350
55	aggregation	4	1	equal	75	replacement	NULL	green_1	exp_gtsrb_20200303-145647
56	aggregation	4	1	equal	100	replacement	NULL	green_1	exp_gtsrb_20200304-062110
57	aggregation	4	1	equal	12,5	replacement	NULL	black_1	exp_gtsrb_20200310-154035
58	aggregation	4	1	equal	25	replacement	NULL	black_1	exp_gtsrb_20200309-173030
59	aggregation	4	1	equal	50	replacement	NULL	black_1	exp_gtsrb_20200309-173225
60	aggregation	4	1	equal	75	replacement	NULL	black_1	exp_gtsrb_20200310-073100
61	aggregation	4	1	equal	100	replacement	NULL	black_1	exp_gtsrb_20200310-221645
62	aggregation	9	1	equal	25	replacement	NULL	green_1	exp_gtsrb_20200303-183227
63	aggregation	9	1	equal	50	replacement	NULL	green_1	exp_gtsrb_20200303-100605
64	aggregation	9	1	equal	75	replacement	NULL	green_1	exp_gtsrb_20200304-174927
65	aggregation	9	1	equal	100	replacement	NULL	green_1	exp_gtsrb_20200304-175046
66	aggregation	8	2	equal	25	replacement	NULL	green_1	exp_gtsrb_20200321-154705
67	aggregation	8	2	equal	50	replacement	NULL	green_1	exp_gtsrb_20200321-011532
68	aggregation	8	2	equal	75	replacement	NULL	green_1	exp_gtsrb_20200320-225526
69	aggregation	8	2	equal	100	replacement	NULL	green_1	exp_gtsrb_20200320-090424

Figure 8.1: Experiment result file overview of traffic sign case study

8.4 Requirements for PySyft

	A	B	C	D	E	F	G	H	I
1	Merge strategy	# benign clients	# malicious clients	distribution	% poisoned da	attack model	order of time	backdoor	name_of_result
2									
3	Experiments on numbers of clients:								
4	sequential	1	0	equal	NULL	NULL	NULL		exp_yale_20200321-132011
5	sequential	5	0	equal	NULL	NULL	NULL		exp_yale_20200317-095112
6	sequential	10	0	equal	NULL	NULL	NULL		exp_yale_20200320-091322
7	aggregation	2	0	equal	NULL	NULL	NULL		exp_yale_20200321-111208
8	aggregation	5	0	equal	NULL	NULL	NULL		exp_yale_20200320-110135
9	aggregation	10	0	equal	NULL	NULL	NULL		exp_yale_20200320-091248
10									
11									
12	Experiments on order of time (of backdoor insertion)								
13	sequential	4	1	equal	100	basic	first	beard	exp_yale_20200317-111217
14	sequential	4	1	equal	100	basic	last	beard	exp_yale_20200317-095227
15	sequential	9	1	equal	100	basic	first	beard	exp_yale_20200321-091703
16	sequential	9	1	equal	100	basic	last	beard	exp_yale_20200323-103255
17	sequential	4	1	equal	100	basic	first	glasses	exp_yale_20200322-202918
18	sequential	4	1	equal	100	basic	last	glasses	exp_yale_20200322-203036
19	sequential	9	1	equal	100	basic	first	glasses	exp_yale_20200322-212541
20	sequential	9	1	equal	100	basic	last	glasses	exp_yale_20200322-212525
21									
22	Experiments on attack model								
23	aggregation	9	1	equal	100	basic	NULL	beard	exp_yale_20200320-173907
24	aggregation	8	2	equal	100	basic	NULL	beard	exp_yale_20200320-182046
25	aggregation	7	3	equal	100	basic	NULL	beard	exp_yale_20200320-190311
26	aggregation	9	1	equal	100	basic	NULL	glasses	exp_yale_20200322-173547
27	aggregation	8	2	equal	100	basic	NULL	glasses	exp_yale_20200322-141859
28	aggregation	7	3	equal	100	basic	NULL	glasses	exp_yale_20200322-173612
29	aggregation	4	1	equal	12,5	replacement	NULL	beard	exp_yale_20200320-150704
30	aggregation	4	1	equal	25	replacement	NULL	beard	exp_yale_20200320-140717
31	aggregation	4	1	equal	50	replacement	NULL	beard	exp_yale_20200320-142106
32	aggregation	4	1	equal	75	replacement	NULL	beard	exp_yale_20200320-125400
33	aggregation	4	1	equal	100	replacement	NULL	beard	exp_yale_20200320-115106
34	aggregation	4	1	equal	12,5	replacement	NULL	glasses	exp_yale_20200322-182416
35	aggregation	4	1	equal	25	replacement	NULL	glasses	exp_yale_20200322-011552
36	aggregation	4	1	equal	50	replacement	NULL	glasses	exp_yale_20200322-182351
37	aggregation	4	1	equal	75	replacement	NULL	glasses	exp_yale_20200322-095535
38	aggregation	4	1	equal	100	replacement	NULL	glasses	exp_yale_20200322-115310
39	aggregation	9	1	equal	12,5	replacement	NULL	glasses	exp_yale_20200322-200923
40	aggregation	9	1	equal	25	replacement	NULL	glasses	exp_yale_20200322-190337
41	aggregation	9	1	equal	50	replacement	NULL	glasses	exp_yale_20200322-190323
42	aggregation	9	1	equal	75	replacement	NULL	glasses	exp_yale_20200322-193415
43	aggregation	9	1	equal	100	replacement	NULL	glasses	exp_yale_20200322-193425
44	aggregation	9	1	equal	12,5	replacement	NULL	beard	exp_yale_20200320-170002
45	aggregation	9	1	equal	25	replacement	NULL	beard	exp_yale_20200320-155935
46	aggregation	9	1	equal	50	replacement	NULL	beard	exp_yale_20200322-000349
47	aggregation	9	1	equal	75	replacement	NULL	beard	exp_yale_20200320-150555
48	aggregation	9	1	equal	100	replacement	NULL	beard	exp_yale_20200320-155216

Figure 8.2: Experiment result file overview of face recognition case study

Listing 8.4: requirements.txt for PySyft

```
flask_socketio~=4.2.1
Flask~=1.1.1
lz4~=3.0.2
msgpack~=1.0.0
numpy~=1.18.1
phe~=1.4.0
Pillow~=6.2.2
requests~=2.22.0
scipy~=1.4.1
syft-proto~=0.2.9.a2
tblib~=1.6.0
torchvision~=0.5.0
```

```
torch~=1.4.0
tornado==4.5.3
websocket_client~=0.57.0
websockets~=8.1.0
```

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org`, 2015. Accessed: 2019-04-28.

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM. doi:10.1145/2976749.2978318.

[AMDJ12] Luis Alvarez, Marta Mejail, Luis Deniz, and Julio Jacobo. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings*, volume 7441. Springer International Publishing, 09 2012. doi:10.1007/978-3-642-33275-3.

[AT19] Mamoun Alazab and MingJian Tang, editors. *Deep Learning Applications for Cyber Security*. Springer International Publishing, 2019. doi:10.1007/978-3-030-13057-2.

[BBB13] Djamel Eddine Benrachou, Brahim Boulebtateche, and Salah Bensaoula. Gabor/pca/svm-based face detection for driver's monitoring. *Journal of Automation and Control Engineering*, 1:115–118, 01 2013. doi:10.12720/joace.1.2.115-118.

[BCL+18] N. Baracaldo, B. Chen, H. Ludwig, A. Safavi, and R. Zhang. Detecting poisoning attacks on machine learning in iot environments. In *2018 IEEE International Congress on Internet of Things (ICIOT)*, pages 57–64, 2018.

[BHN+18] Josef Bengtson, Filip Heikkilä, Per Nilsson, Lukas Nyström, Erik Persson, and Gustav Tellwe. Deep learning methods for recognizing signs/objects in road traffic. `https://api.semanticscholar.org/CorpusID:55655029`, 2018. Accessed 27-December-2019.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1175–1191, New York, NY, USA, 2017. ACM. doi:10.1145/3133956.3133982.

[BK18] Shreyas Bhat and Sudeep Katakol. Federated machine learning on blockchain. `https://github.com/shreyasnbhat/federated-learning`, 2018. Accessed: 2019-04-28.

[BNL12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1467–1474, Madison, WI, USA, 2012. Omnipress.

[BVH+18] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the 23nd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 3-5 June 2020, Palermo, Sicily, Italy*. CoRR, 2018, abs/1807.00459.

[CCB+19] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019*, volume 2301 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[Dan20] Dana Foundation. Diagram of a neuron, 2020. URL `https://www.dana.org/article/cells-of-the-brain/`. Accessed: 2020-01-23.

[Dec18] DecentalizedML. Decentralized machine learning whitepaper, 2018. URL `https://decentralizedml.com/DML_whitepaper_31Dec_17.pdf`. Accessed: 2019-04-28.

[Fac19] FaceApp Inc. FaceApp. `https://www.faceapp.com/`, 2019. Accessed 2019-10-07.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas 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, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2810103.2813677.

[GB12] S. Gudivada and A. G. Bors. Face recognition using ortho-diffusion bases. In *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 1578–1582, 2012.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts ; London, England, 2016.

[GDG17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proceedings of the Machine Learning and Computer Security Workshop*, Long Beach, California, USA, 08 Dec 2017.

[Goo20] Google Maps. Photo of Rainergasse crossing Blechturmgasse in Vienna. `https://goo.gl/maps/PXggTjjJ57rrLELK6`, 2020. Accessed: 2020-04-29.

[GPM16] Ian J. Goodfellow, Nicolas Papernot, and Patrick D. McDaniel. cleverhans v0.1: an adversarial machine learning library. *CoRR*, abs/1610.00768, 2016, 1610.00768.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations 2015, ICLR 2015*, 2015.

[HRM+04] Alan Hevner, Alan R, Salvatore March, Salvatore T, Park, Jinsoo Park, Ram, and Sudha. Design science in information systems research. *Management Information Systems Quarterly*, 28:75–, 03 2004.

[IMA+16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016, 1602.07360.

[JH19] Gretchen Jackson and Jianying Hu. Artificial intelligence in health in 2018: New opportunities, challenges, and practical implications. *Yearbook of Medical Informatics*, 28:052–054, 08 2019. doi:10.1055/s-0039-1677925.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KMA+18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3390–3398. AAAI Press, 2018.

[KMRR16] Jakub Konecný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016, 1610.02527.

[KMT13] Hurieh Khalajzadeh, Mohammad Manthouri, and Mohammad Teshnehlab. Hierarchical structure based convolutional neural network for face recognition. *International Journal of Computational Intelligence and Applications*, 12, 09 2013. doi:10.1142/S1469026813500181.

[KMY+16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[KNH12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny. *University of Toronto*, 05 2012. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi:10.1073/pnas.1611835114.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LAKG98] Michael Lyons, Shigeru Akamatsu, Miyuki Kamachi, and Jiro Gyoba. Coding facial expressions with gabor wavelets. In *Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition*, volume 1998, pages 200 – 205, 05 1998. doi:10.1109/AFGR.1998.670949.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[LDG18] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. *CoRR*, abs/1805.12185, 2018, 1805.12185.

[LXS17] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48, 2017.

[MDFFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2016.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[MSCS18] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2018.

[NRMS20] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning attacks on federated learning-based iot intrusion detection system. In *Workshop on Decentralized IoT Systems and Security (DISS) 2020*, San Diego, CA, United States, Feb 2020.

[NST+18] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.0.1. *CoRR*, 1807.01069, 2018.

[PAH+17] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13:1333–1345, 2017.

[PNBK97] J. Hespanha P. N. Bellhumer and D. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Special Issue on Face Recognition.*, 17(7):711–720, 1997.

[PSS07] Thomas Pedersen, Yucel Saygin, and Erkay Savas. Secret sharing vs. encryption-based techniques for privacy preserving data mining 1. 12 2007. URL `https://api.semanticscholar.org/CorpusID:17401136`.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[RGR18] David Reinsel, John Gantz, and John Rydning. The Digitization of the World, from Edge to Core - An IDC Whitepaper, 2018. URL `https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf`. Accessed: 2019-04-28.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, American Psychological Association (APA)*, 65(6):386–408, 1958. doi:10.1037/h0042519.

[RTD+18] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. In *Privacy Preserving Machine Learning NeurIPS 2018 Workshop*, Montréal, Canada, Dec 2018.

[SAK+19] Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef, and Itai Zeitak. Overcoming forgetting in federated learning on non-iid data. In *2nd International Workshop on Federated Learning for Data Privacy and Confidentiality at NeurIPS 2019*, Vancouver, Canada, Dec 2019.

[SGR+19] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi. Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 270–274, 2019.

[SKL17] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 3520–3532, USA, 2017. Curran Associates Inc.

[SKSM19] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can you really backdoor federated learning? In *2nd International Workshop on Federated Learning for Data Privacy and Confidentiality at NeurIPS 2019*, Vancouver, Canada, Dec 2019.

[SRE+19] Micah J. Sheller, G. Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In Alessandro

Crimi, Spyridon Bakas, Hugo Kuijf, Farahani Keyvan, Mauricio Reyes, and Theo van Walsum, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 92–104, Cham, 2019. Springer International Publishing.

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1310–1321, New York, NY, USA, 2015. ACM. doi:10.1145/2810103.2813687.

[SSSI12] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323 – 332, 2012. doi:https://doi.org/10.1016/j.neunet.2012.02.016.

[SSSS17] 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)*, pages 3–18, May 2017. doi:10.1109/SP.2017.41.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, August 2016. USENIX Association.

[WYS+19] 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 *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, May 2019. doi:10.1109/SP.2019.00031.

[Xl19] Baidu X-lab. Advbox:a toolbox to generate adversarial examples that fool neural networks, March 2019. URL `https://github.com/baidu/AdvBox`. Accessed: 2019-12-03.

[YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2):12:1–12:19, January 2019. doi:10.1145/3298981.

[ZLS+20] Haoti Zhong, Cong Liao, Anna Cinzia Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, CODASPY '20, page

97–108, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3374664.3375751.