University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Byzantine Robust Aggregation Mechanism for Decentralized Federated Learning

Michael Vuong
Zurich, Switzerland
Student ID: 16-925-950

Supervisor: Chao Feng, Alberto Huertas
Date of Submission: July 14, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Föderales Lernen wird immer populärer, da die herkömmlichen Methoden des maschinellen Lernens immer mehr an ihre Grenzen kommen. Mit dem technologischen Wandel können die heutigen Geräte wie mobile Telefone, immer mehr Daten speichern. Um diese kostbaren Daten nutzen zu können, müssen immer mehr Daten transferiert werden. Zum einen verletzt dies die Datenprivatsphäre, da nicht jede Person seine persönlichen Daten teilen möchte. Zum anderen ist es nicht immer möglich die Unmengen von Daten zu transferieren. Um dieses Problem entgegenzusetzen ist Föderales Lernen entstanden. Mit dieser neuen Technik lässt sich die erwähnten Probleme umgehen, da keine Daten Transfer mehr benötigt wird. Es gibt zwei Hauptmethoden, die sich bisher bewährt haben, ein zentralisierter und dezentralisierter Ansatz. Zum zentralisierten Ansatz gibt es schon viele Methoden, da diese auch sehr beliebt und oft genutzt wurde. Der andere Ansatz hingegen, blieb bisher noch im Schatten. Diese Arbeit zielt daraufhin, eine neue sichere Aggregationsregel zu finden, welche im dezentralen Ansatz funktioniert, um die Forschung in diesem Gebiet voranzutreiben. Dies geschieht durch eine Erweiterung auf einem existierenden Framework fedstellar, welches den dezentralen Ansatz simuliert oder sogar ausführt. Die Evaluation des Algorithmus hat gezeigt, dass dieser situationsabhängig ist und in manchen besser funktioniert als anderen. Des Weiteren wird auch gezeigt, wo die Grenzen sind und wie man diesen Algorithmus auf andere Plattformen erweitern kann, um seine Performance weiter zu testen.

Abstract

Federated learning has become increasingly more popular due to limitations of the traditional machine learning methods regarding the data privacy. In addition due to technological evolution, the data volume in general has increased by a lot. Mobile devices are capable of storing more and more data. While traditional machine learning methods struggle to deal with these concerns, federated learning emerged from these problems. Two main approaches have been mainly used namely Centralized and Decentralized Federated Learning. The former one has gotten much more attention in comparison with its counterpart and thus possesses many aggregation rules which are resistant to attacks. The goal of this thesis is to propose a new aggregation rule which is resistant to attacks against the machine learning model for the decentralized setting to fill a gap where the research has no reached yet. This is done by extending an existing framework fedstellar, for federated learning. The case studies as part of the evaluation evaluate the algorithm on performance and resource consumption related metrics. They indicate that the performance of the algorithm depends on the situation. They also show the limitation of the algorithm and possibilities of expanding the algorithm to other applications.

Contents

Chapter 1

Introduction

”Traditional” machine learning methods usually rely on centralized structure where data is collected from various sources and then transferred to a central server. After the data transfer typically a single model on the central entity is trained on the data set. While these methods have proved to be effective, they ignore issues like data privacy. Federated learning (FL) is a newer approach to machine learning. In contrast to traditional machine learning where data is centralized, it is retained by the participant instead and not shared with anyone else by relying on a new way to handle data. Instead of transferring data to a central entity, the training and aggregation process is done by the participants where data is collected. This approach has several advantages over ”traditional” approaches.

First, as mention before, data privacy is violated by transferring data, because in many cases the data is sensitive and not desired to be shared. In FL, the data remains on the participants side and only the model updates which are calculated after local aggregation, are being shared. This preserves data privacy issues, because the raw sensitive data is not being exposed anymore. Second, another challenge with data transfer are the data volumes. Raw data can get up to higher numbers really quickly which is why it is not always possible in the first place to transfer data. FL allows the use of data without having to transfer them. Furthermore depending on the data volume, a data center is needed for the training process, as no single device could handle such huge amount of data.

Thanks to these properties, FL is getting increasingly more popular. A new opportunity also comes with new danger. Because of its distributed nature, it is vulnerable to specific attacks, especially poison attacks. It is also vulnerable to byzantine faults. To mitigate the impact of such attacks, a proper defense in form of an aggregation mechanism or rule is needed. It means that before the update parameters are being aggregated to compute a new model, some precaution measures have to be taken, so that the new model is no being affected by malicious clients.

1.1 Motivation

FL has gained a lot of attention in recent years. Since its birth date, Centralized FL (CFL) has always been commonly used. In CFL a central entity manages the whole training process. The local devices perform their training on their respective local data and sent their updates parameters to the central sever which then aggregates and updates the global model. This process is repeated many times. A centralized approach is often vulnerable to a single point of failure where for example the central entity can malfunction. To address these concerns, Decentralized FL (DFL) has come into the picture. In DFL there is no central entity managing the training process and aggregation. Instead the participants exchange the update parameters with themselves, more specifically their neighbors. Despite its relevance and advantages, the field of DFL has not advanced far enough especially regarding the security and robustness.

With a variety of environments using DFL, the need for more secure aggregation has risen as well. This is because different aggregation work better on different environments. This is where this thesis focuses its work on. By comparing multiple aggregation mechanisms and their usage, a new aggregation is going to be proposed, where it fits the scenarios where other aggregation have not covered yet.

1.2 Description of Work

This thesis aims to propose a new aggregation mechanism for DFL. It should have several properties to resist byzantine attacks. To achieve this goal, this thesis is split into several sections. First section revolves around the literature research. First an introduction to machine learning and its most important concepts are being talked about. This is particularly important for knowing the difference between FL and traditional machine learning with its strength and weaknesses. The focus lays on the weaknesses because the understanding of them are needed for developing of a defense. Different aggregation mechanisms are going to be compared with each other on their attributes such as their robustness overall, usage, environment they are being most effective in. Next, an existing framework for DFL has to be analyzed on its functionality in order to gain knowledge to develop a new aggregation later on this framework. Finally, a new aggregation is going to be proposed. This includes a theoretical view on how this algorithm works and satisfies all the condition, a pseudo code for a better understanding on what the algorithm does and the implementation and changes themselves on the framework. Furthermore the new algorithm will be evaluated on different setting criteria which will be discussed in the evaluation chapter.

1.3 Thesis Outline

The thesis is going to be structured as following. The second chapter serves as an introduction to machine learning as well as to the more specific topic federated learning.

Because federated learning is a topic within the machine learning field knowledge about it must be obtained first. This thesis will start with an overview of machine learning, then move on to data processing to understand what what attributes data has in machine learning and what labels are. The next will be about supervised and unsupervised learning where classification, regression types in machine learning and also gradient descent algorithms which are important in this thesis are being looked at. Last this thesis will tackle neural networks because this is also something important for this thesis.

After the background chapter where the necessary knowledge have been gathered, this thesis will move on to works which have been done on federated learning topics which are relevant to our thesis. These include byzantine robust approaches, poison attacks on federated systems, several techniques to improve robustness.

The chapter after that will be about the implementation. The implementation is done on a framework. This means that some part of the code will change and this thesis will discuss what changes have been made and what their impact is. It will also explain the whole idea behind this in detail. Starting of with the rough idea and then going step by step more into detail.

As seen above, the implementation is done on a framework. In this chapter this thesis will evaluate the implementation. By creating several case studies, the most important factors can be examined. In each of the case studies, one of the factors will be chosen to be evaluated on. This means that the other factors are being ”fixed” and vary the chosen factor to see the differences. This will be repeated for each factor. CPU usage, performance, accuracy and more performance and resource related attributes are being taken into consideration. The conclusion and summary of this work will end the thesis.

Chapter 2

Background

With the rise of chatGPT [1] the word machine learning and artificial intelligence are a hot topic right now. But fewer understand what machine learning actually is. Thus the concepts of machine learning have to be understood first, before moving to FL. After knowing roughly how machine learning works, an introduction to FL is provided to understand the differences between FL and traditional machine learning in particular their strength and weaknesses. The architecture is also being mentioned. Afterwards the two main approaches in FL, namely Centralized Federated Learning and Decentralized Federated Learning, are being explained. Lastly a short view on the main weakness of FL is being shown.

2.1 Machine learning

2.1.1 Introduction

Machine learning [2] is as its name suggest is a field in where revolves around understanding and building methods that let machine learn, on specific tasks with the help of data to improve their performance over time. The key point is being able to learn without explicitly being told to do so. The following example shows an environment which machine learning can be used in. There is a game called "rock-paper-scissors". In this game the objective is to mimic these following three objects with the hand. A full open spread hand represents a paper. A fist equals a rock. Putting two fingers up and the rest down would be scissors. The rules are that rock beats scissors, scissors beats paper and paper beats rock. For example, paper wins against rock. Now without machine learning, to identify scissors, the computer must be told exactly how it looks like. This could be something like this.

```
If 2 fingers up and 3 fingers closed then its a fist
```

First of all, this is a very vague description. What is a finger? How does the computer know how they look like? So maybe it would be better to describe the silhouette of the

scissors in terms of coordinate. This could be one condition for scissors. The problem is that the computer would have to add new rules every time they differ from the current model of scissors. For example the skin color, size etc. would be factors to consider.

Machine learning aims to solve this problem by adding those new rules by itself without be told to. To be able to do this, a machine learning model usually needs data. A plethora of pictures of scissors can be the data for the machine learning model. To emphasize, machine learning is, to program computers to optimize performance by using example data, such that they can predict or "make their own decisions". Machine learning algorithms aim to build a model based on example data, also now as training data, because it conducts its "training (learning)" on that data. The goal is that these models can do decisions or predictions without specifically having us to tell them to do by changing the code. They also improve as time and training cycle go on. So for this example, in the end the expectation is, that the computer can recognize the hand gestures.

There are several types of machine learning such as supervised learning, unsupervised learning. But first, a closer look at how data is managed is provided in the next section.

2.1.2 Data in Machine Learning

Data is a key component in machine learning. They come in many different forms such as numerical, time series, information [3]. It contains of sets of observation of past events or experiment which can be used to train the machine learning model. They can be collected using techniques such as measurement, observation and analysis. The quality of data is an important factor which can determine the outcome of a learning process of a machine learning model. "Bad" data can lead to results which are not meaningful. Machine learning models use data to find patterns and relation between inputs and outputs. With more iteration passing of this process, they learn and can be used for various purposes.

Data is usually divided into two different categories; Labeled Data [4] and Unlabeled Data. Labeled data is a pair of a data point and a target variable (label). The model tries to predict the result and can look it up if it was correct or not with the help of the label. A picture of a banana and the word "banana" together would be an example for labeled data.

Unlabeled data does not have a label so it only contains a picture in the case above. When thinking about machine learning data, usually the first thing to come into mind is the example above. The second thing is mostly a series of numbers like 2,3,5,8,12,17,.. in which the model should predict the pattern. Those are the two most typical data form called categorical for the picture example and numerical for the series example. Numerical data usually contain values that can be sorted. Few examples are weight, height, age. Categorical data contain values that represent a category such as gender, sex, age group, education level. Figure 2.1 shows an example of labeled data.

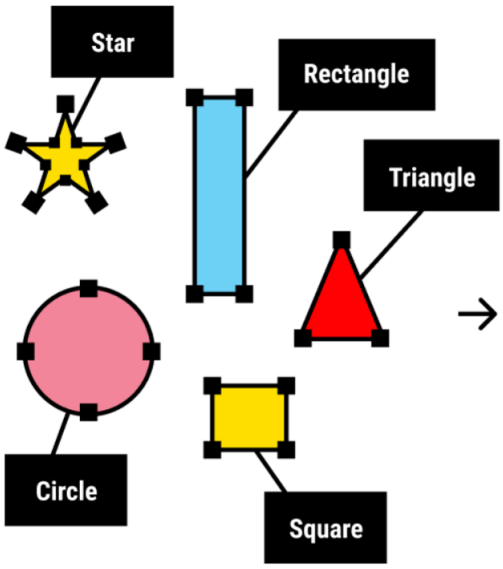

Figure 2.1: Objects with their respective label

When using data for machine learning, the data is split into three sets. The first one is the training set which is used to feed the model for the training phase. This is done repeatedly. One training round is called an epoch. It consist of the machine learning model going over the entire provided data set (here the set used for training). It is important to note that the same data is used in each iteration. Training set should be diverse in terms of input so the model can cover a lot of ranges and scenarios that might appear. The goal is that it can predict any unseen data in the future. Obviously this is not going to be possible for 100 percent of the time, but the goal is to cover as much as possible.

The second set is the validation set. This set is used to evaluate during the training. It indicates, if the model is going into the right direction. After each epoch the evaluation will be done on the validation set. By doing this the models hyper parameters are being tuned. The main goal is to prevent over fitting which is the case when the model can make accurate decisions on the training data set but not on unseen data.

The last set is the testing set. This data set is used after the training has been completed and serves as an test. It is unseen data an acts as an last instance to answer the question on how good our model performs.

A machine learning model's training route is similar to how humans learn. An example of a course at an university shall demonstrate the process. In school, a new subject is being taught, for example new technique in math. To strengthen the knowledge about the new learned principles, usually weekly exercises have to be done on that topic. This is similar to the training set and validation set cycle which. One epoch would be solving exercises. At the end of a semester an exam takes place, which would be a equivalent of the evaluation on the testing set. Just like the testing set, new questions which have not seen before (unseen data) are being asked and by applying the learned knowledge, the problems can be solved.

Data has to be pre processed first before its use. This means preparing the data to be used. ”Bad” data quality can lead to bad results. So it is very important to have good data. Pre processing data includes normalizing data, handling missing values and filtering (not suitable) values out.

2.2 Supervised Learning

Machine learning can learn from past experiences based on data which is being fed to the model. For example, consider the rock-paper-scissors game . The machine learning model should predict if a picture provided is rock, paper, scissors or none of them. The model will look up past experiences which in this case are pictures it has seen before. According to the features, it will decide if the new picture matches one of the three signs or not. As discussed before a rock would be represented by something like a fist. This is the basic concept of how machine learning ”learning” process works. Machine learning models have to train on data set in order to be able to make prediction like this. There are various methods on how to train. The first one is called supervised learning [5]. With this method labeled data is used to train the model. Labeled data consist of a pair of input and output data. For example, a picture of a rock, and the string ”rock” together.

The machine learning model tries to learn the relation between the input and output so it can predict the output in the later stages of the training process based on the input data. The input consist of features, attributes or characteristics of the data. Based on those criteria the model has to make a decision. Output data is basically the answer to the input data. It can be simply a string or other targets that the algorithms and the models tries to predict. Supervised learning can be divided into two categories.

2.2.1 Classification

As the name already tells us, classification is a process of dividing things into categories, by a machine. [6] So for example, by holding an apple before the camera, the goal is, that the computer should be able to tell, that it is an apple. This can also work with other items such as recognizing pictures and other objects. Classification is basically the problem of identifying things. Categories are for example, ”fruits”, ”blood-type”, ”size”(small, medium, large). The machine learning model trains on a data set containing many items which are labeled with the category they belong to. Then it tries to categorize them. This process is repeated many times until a certain amount of items have been identified correctly. If the machine learning model is accurate enough, in the last step it can be tested on unseen data to see if it actually is working. For example, assume pictures with cats and dogs on it. They are also labeled with ”cat” and ”dog” respectively. For each of the pictures, the task now for the machine learning model is, to put them either in the category ”cat” or ”dog”. After each epoch (after all pictures have been categorized), an evaluation will be done to see, how many of them the model got correct.

The example above only has two categories. This is called binary classification. If there are more than two categories then it becomes a multi class classification. For example,

blood type would fit into multi class classification because there are more than two blood types. Some of the relationship between input and output are really simple. So simple that there is a linear relation between them. These linear classification are computationally efficient. Non-linear classification are more complex and the relation is not linear.

Classification learners can learn during or after the training phase. Lazy learners learn after the training phase and only store training data and to nothing else with it. When the testing phase arrives, they will use their technique(e.g. k-nearest neighbors) to return a result. This is also the phase where they actually learn and update their model. With this method, the training time is significantly shorter than usual as they only store data. A eager learner in comparison learns during the training phase, meaning they update their model during training. This makes it such that the training phase takes much more time as more computation have to be done. But during the testing phase a eager learner will return the result immediately as it already has learned and can apply its model to the new data as soon as possible.

After all the training, some values or guidelines on how our model performed is needed. To do this evaluation is needed. In this step different metrics are used in order ton measure the performance of the model.

Following metrics could be a possible parameter for evaluation:

- Accuracy is the percentage of correct identified instances
- Confusion matrix is a table that shows true positives, true negatives, false positives and false negatives
- Precision and recall: precision is true positive over total number of all predicted positives including those not identified correctly, where as recall is true positives over total number of actual positives
- F1 score is an accuracy metric, the harmonic mean of precision and recall
- Receiver Operating Characteristic curve and Area Under the Curve are plots for performance
- Cross validation is dividing data into subset and training several model on it evaluating them against each other

2.2.2 Regression

In contrast to classification, regression aims to produce numbers like height, age and income amount. Regression finds the relation between the output variable and one or ore input variables [7]. The probably most known one is linear regression where one finds a line which fits the data set. By doing this, an estimate of a trend or rule of the data points can be found. Using the result the model can predict outcomes with it. A popular application of this is in the finance world. Assume that the market index is m where the movement of the market index is shown. Also assume that a fictional firm a which has

its own index n. Now it would be interesting to see how the index of the firm a moves in contrast to the market index. By doing linear regression, the relation between those two index can be calculated. Figure 2.2 shows an example of linear regression.

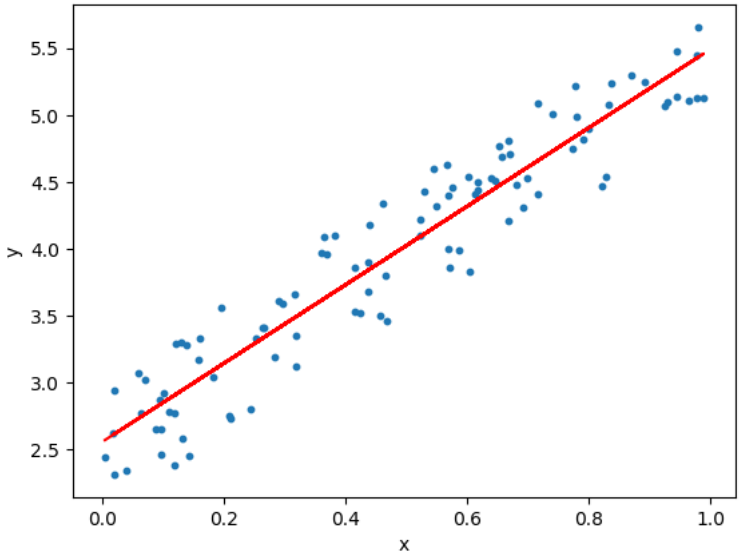

Figure 2.2: Linear Regression: A line through the data which indicates the trend of the data points

There are other regression types which will not be covered in this thesis.

2.2.3 Gradient Descent

With the knowledge on how important the accuracy and performance of the model is and how to measure it, adjustments need to be done. These adjustment are needed to improve the model. This can be done by applying a so called gradient descent on a loss function of the model. First, a loss function is a function which indicates the difference between the value of our model and the actual true value. The number gets increasingly higher the less accurate a model is. A loss function could look like this:

$$Loss\ function\ (L) = \frac{1}{n} * \sum (actual - predicted)^2$$

This is an important function as it tells us if our model is going into the right directions. By changing the model parameters, the function indicates, if the changes are actually doing anything. By using the loss function the performance changes can be measured, with each change of model parameters.

Gradient descent is basically the procedure of doing the above. A gradient is a vector which points in the direction of the largest increase of value per step. [8] This means going into this directions will eventually lead us to a local minimum or maximum. The

loss functions indicates how well the model performs. The higher the number, the worse it is are doing. So by finding the gradient in the loss function it knows in which direction it should not go. Instead it goes in the opposite way of the gradient which gives this method its name. These are the following steps for gradient descent:

1. Initiate the parameters randomly
2. Calculate the gradient of the loss-function after each epoch of training
3. Walk the opposite way of the gradient and update the parameters accordingly
4. Repeat 2 and 3 until meaningful results have been gathered

As this is not a mathematical thesis the mathematical background and proof of this method will not be covered . This is just a theoretical point of view.

2.3 Unsupervised Learning

Learning methods with labeled data has been explained. They act similar to solutions provided to exercises which can help to see if the exercises were solved correct or not. However unsupervised learning does not have labeled data. So there is no more guide or help for that matter. Instead finding the solutions is the goal here with out the label. Consider the example with cat and dogs. This time the machine does not know anything about cat or dogs. But it can differentiate them according to their features.

Figure 2.3: Various types of dogs and cats. Even plush toys dogs can be identified

A human can easily differentiate the objects shown in figure 2.3. He or she would divide the five animals into two different categories. Note that the person also did not have to learn anything before but still could differentiate them. So in this case the person had to work on its own, to detect features and pattern to sort them in order to categorize the things. [9]

Unsupervised learning is also divided into two main categories similar to supervised learning. The first one is clustering. As the name suggest, it is a method where features are being divided into groups according to references which they may have. In general it

is used to bring some structure into the non-labeled data, by grouping (clustering) attributes. For example, following picture displays three main areas where the data dots are. So by nature, clustering them together makes most sense, meaning there will be three clusters at the end. Figure 2.4 shows three data heaps being grouped into three clusters.

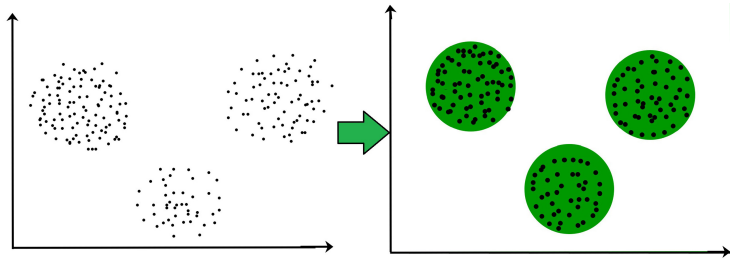

Figure 2.4: Our natural eyes cluster them by instinct, because they are so close to each other

Clustering has no rule or strategic plan on how to do it. It depends on the user or machine. For some cases there are more than just one way to cluster things. Thus a machine learning algorithm has to make assumption in order to do the clustering process. There exist multiple variation of clustering. There is density based clustering which is the example above. Density based ones tend to cluster the region or areas with most data points in them together. They have good accuracy and can merge two clusters if needed to.

Hierarchical Based Methods form a hierarchical tree where groups are clustered. An example would be dividing customers into groups based on their income.

Partitioning methods divide objects into several clusters. Each partition typically forms a cluster. A very popular method is the k-means clustering. It is one of the simplest algorithm where it simply assigns the data points into clusters based on their nearest cluster which is calculated by the mean of the local data points.

2.3.1 K-Means Clustering

A concrete method for unsupervised learning is the following. K-Means aims to divide the data into several groups where k indicates the number of groups it creates in the process. These groups are also called clusters. The goal of such clustering is to make the data points from each group more comparable to the other groups. It is basically a grouping based on how similar or different they are from each other. [10] Assuming a data set with data points. To calculate the similarity for grouping, the euclidean distance is a possible way to do so. Then proceed as follow

- First randomly initialize k points. They act as the mean or the ”central points” for each of the k clusters

- Then categorize for each data point closest to one of the k points and update the mean of them. So at the beginning, there will be only one point which is the k point itself. In the second iteration add another data point to the cluster and update the mean of the cluster
- Repeat this process then until no more data points are available

As the methods name already suggest the k points are called mean because they are the mean of all the data points allocated to them. There are multiple ways to initiate them at the beginning, not only random as before. But simplest way is to just randomly initiate them. So in this case choose k data points and mark them as k points.

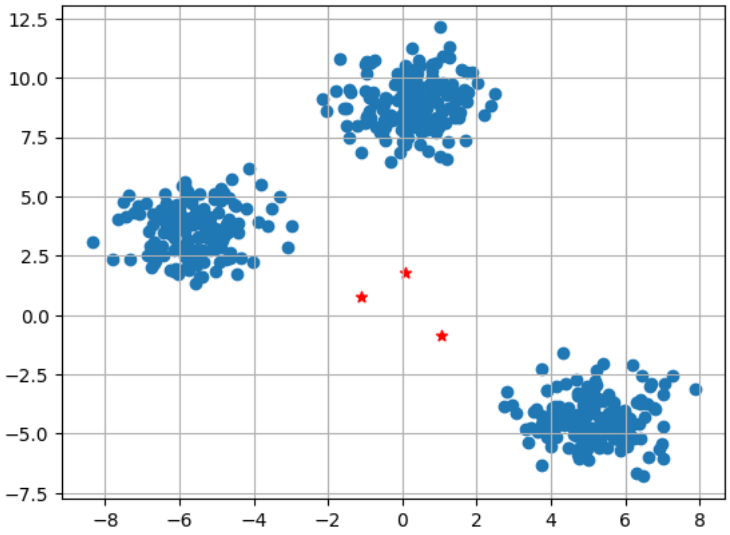

Figure 2.5: Three red stars as k points initiated in the center

Assume that the initiated marked points are k points as indicated in the figure 2.5. This is the initial state. The k stars are randomly being created in the center. To solve this problem the euclidean distance is used again. If the distance from a data point to the k point is greater than a threshold then consider this a new k point. Figure 2.6 shows the situation after the algorithm has finished.

The mean changes overtime and at the end of the clustering they are in the right position as their name suggests.

Alternatively, set the condition that the k points have to be x distance away from each other. But this would require us to vaguely know the topology of the data points. The second way to initiate them is to select points in the boundaries of the data set. In this case it does not necessarily have to be a data point in the data set.

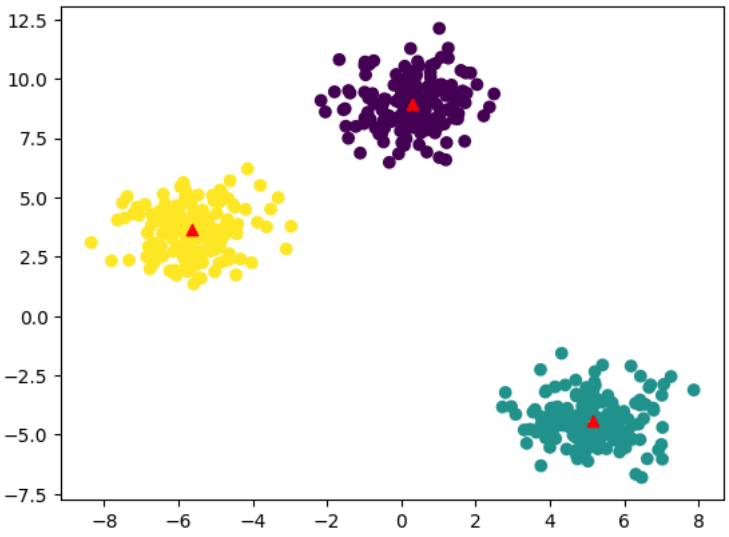

Figure 2.6: End position of the k points

Algorithm 1 shows how a k-mean initialization can look like. Algorithm 1: K-Mean Clustering

```
K–Means Clustering initialization

Select k random data points and mark them as k points
For each data point in data set:
    Find closest k point to the current data point
    Add current data point to that cluster
    Update the mean of the cluster
End for
```

2.4 Reinforcement Learning

Reinforcement Learning is a method where the machine learns how to maximize the reward or minimize the penalties given, for a specific scenario.[11] It is different from supervised learning as there are no key answer attached to the problem but instead an agent that decides on what to do next. To illustrate this consider following situation. A person is put in a maze like structure where there are several ways leading to an exit. But some of the paths are littered with hazards meaning he has to suffer ailments to get through like fire. So a possible solution would include the path through fire but there are other paths with less pain to get through. This is where the agent of the reinforce learning has to make decisions which path it will pick. By trial and error it can identify those hazard as something negative. So it would try to avoid them as much as possible. Reinforcement learning is all about decision making. Similar for machines, they also try to optimize their decision making to maximize the reward for a given scenario. The main difference between reinforcement learning and the other two above (Supervised and Unsupervised Learning) is that data is not part of the input but instead accumulated from trial and error methods

(e.g. walking through fire equals bad). After each step the machine learning algorithm makes it has to evaluate if the step was good, bad or neutral. It is basically an automatic system which can learn without human guidance.

2.5 Dimensionality Reduction

Data is huge in the field of machine learning. It is not only big but may be high dimensional too. To handle or work with complex data is not always desirable or possible. High dimensional data is computationally hard to deal with and thus aiming to reduce the dimension of the data to a certain amount where the core information of the original raw data is sill preserved , but gaining the advantages of lower dimensional data, is preferred. Dimensionality Reduction is the process of said reduction of dimension while still retaining as much original information as possible. [12] A dimension in a data set can be a feature or variable. For example speech recognition and signal processing do have large amount of variables and observations. In machine learning the reduction of dimensions can lead to better performance and reduction of complexity of a model. It is also easier to visualize data as it is difficult to display higher dimensional data. Higher dimensional data have the common problem called ”curse of dimensionality” which are problems that only occur at higher dimensions but not in lower ones. In machine learning the complexity of the model worsens quickly in such situations as the number of dimension increases. Thus the reduction of the dimensions to a feasible amount to work with needs to be done There are two main approaches for dimensionality reduction.

First one is feature selection. Here a subset of relevant features of the data set is selected. This is done for various reasons, for example, for complexity reasons as less feature means that it is easier to understand or to lessen the training time of the machine learning model. The main assumption in feature selection is that there are some feature which are simply irrelevant or useless for us. Thus, they can be removed without losing too much information and thus preserving the core information. It is important to note that this method simply returns a subset of the features already being in the data set.

The second one is feature extraction. In contrast to the one above, feature extraction creates new features by combining, changing or transforming the original features in the data set. Some features can be combined into one feature, others can be transformed such that they fit the new data structure and fit the machine learning model as well. A good example is a classic e-mail filter. The machine learning model has to decide if an e-mail is a spam or not. An e-mail contains large amount of features such as title, content, pictures and signature. Some of them may overlap. That is why combining them to reduce the complexity instead of analyzing each of them separately is an option.

2.6 Neural Networks

When thinking about machine learning the term ”neural network” pops up quite often. These artificial neural networks are based on biological neural networks. An artificial

neural network is a collection of nodes(neurons) which very vaguely represent the biological brain. Each connection between the nodes equals the synapses in a brain. They can sent and receive signals to and from other neurons. The connections between neurons are called edges just like the connections in a graph. [13] They have a graph like representation where it consists of nodes and edges. Typically nodes have weights which change overtime as the training proceeds. The weights affect the strength of the signals emitted. A neural network consist of an input layer, one or more hidden layers and an output layer. Each node has a weight as mentioned before and a so called threshold. A node will be activated if the output of that singular node is above the threshold. An activation will cause it to sent a signal to its connecting nodes. Otherwise nothing will happen meaning no data will passed to the next layer as long as the threshold is not reached. How does a neuron work?

The following network has 3 neurons A,B,C and connections between A and C (q) and B and C (r) as shown in figure 2.7.

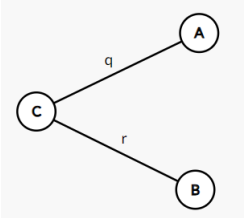

Figure 2.7: C is the input and the outputs are A and B

For this topology the input would be q*A + r*B and the output would be the application of the activation on the input, Function(input). Neural network also rely on data to improve their accuracy over time, but once they finished their training they become really powerful. They allow us to do classification and cluster (supervised and unsupervised training) at very high speed. One of the most known neural network is actually the google search engine.

2.6.1 Example of a Neural Network

The scenario is that a person has to predict if he should go on vacation or not in times of the pandemic season. The final decision should be a one or zero depending on the output, where one means he is going on vacation and a zero means he is not going on vacation. Following factors have an impact on the decision making:

1. Is the country he wants to go to heavily infected or not? (yes: 0,no: 1)
2. Is he vaccinated? (yes: 1, no: 0)
3. Are the regulations regarding the pandemic strict or not? (Yes: 0, no: 1)

Consider following weights:

- Weight 1 = 2, since he does not care about it too much
- Weight 2 = 7, since he thinks that this probably has the largest impact on his decision making
- Weight 3 = 4, since he does not really want to deal with the regulations at the airport and in the target country, but rather focus on vacation

Finally some threshold has to be set, in this case it is set to 8, meaning a bias of -8. After putting in everything in the formula it the result is: (0*2) + (1*7) + (0*4) - 8 = -1 Since 0 > - 1, this means that it is a negative decision. This means even though he is vaccinated but because the country is heavily infected and the regulations are strict, he does not decide to go there. For other people this decision could be different depending on their bias and weights. That example was only for illustration. In reality the neurons have sigmoid-like functions which always have values between 0 and 1 and not strict 0 or 1.

2.7 Federated Learning

Machine learning requires huge amount of data for their models to train on. This is why all the processes to handle data like dimensionality reduction are needed in the first place. In traditional machine learning data is gathered from the participant. This happens everywhere and all the time. For example surfing on the internet an visiting websites. There are many websites that have tried to display ”personalized” adds. A person uses google search to travel the web. As he goes through our live and use the internet, google remembers where he went. Thus it generally knows what he prefers, or at least which websites he prefers to visit. According to the history then, the adds by google have access to our history and try to create adds based on them. So if he often visits sports websites, whether it is for watching soccer games or shopping, the adds will be sport-themed. Now how is this data huge , because after all its only the website google needs to remember. First, it is not only us that is using the internet. So many people nowadays use google search engine. This alone would be large already. But now factoring in that google does not only remember the website's name but it has to add tags to it, because a name alone does not help a machine learning model. For example there are websites , whose name does not reveal their intention at all. So tags like, ”shopping”,”sports”,”education” etc. help later on. This is also called meta data. After the data collection it will be transferred to a central entity usually a server or data center in case of huge data, which will then analyse the data gathered, to get a benefit out of it. As mentioned a transfer of data is required for the entity is going to compute a new model. Data transfer is not always desired or feasible.

The following scenario shall illustrate the problems. A person is working in a car industry and his job is to gather all the sensory data from the cars produced by your company to analyse them such that the cars produced in the future can profit from the results of the analysis. He intends to conduct an analysis each month. In order to do so, he would need to get the sensory data from all the cars of our company. This is the point some problems can be detected already. First the amount of data he would have to transfer is huge. Sensory data of a whole month of a car alone could be huge already. Now imagine each car from our company transferring their data. Second, not every customer is willing to send their data. Maybe they do not want him to know in which places they went with their car. The point is privacy. They would have to exposed their data to him for his analysis. But how can he use those valuable data without violate the privacy issues?

Because of mentioned problems Federated Learning (FL) has been born. FL does not require data from the participants which is the critical point in the situation before. Instead the participants do the analysis of the sensory data themselves. Imagine if the car could create a report each month which includes the most important things, for example the total distance. Privacy has been respected more with this approach because there would be no sharing of raw data but a ”summary” of it instead. This also solves the problem with the data amount. A ”summary” is always less than its raw version.

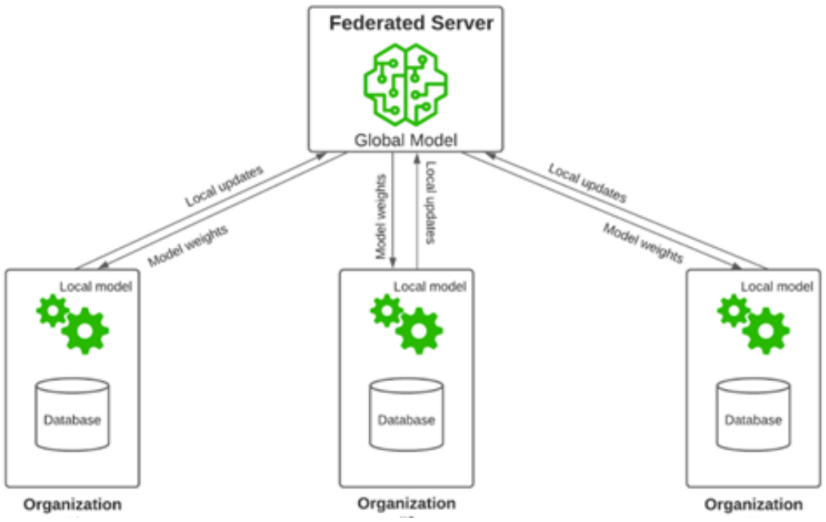

Figure 2.8: The participants do no share their raw data anymore, but their update parameters of their model

FL is a newer approach to machine learning. It aims to be able to scale which it does by using data in privacy preserving way. Without transfer of raw data, the volume has been reduced significantly which makes it suitable for larger scaled projects. [14] When building machine learning models, data is collect and transferred to a central server so the training process can begin. With all the data in one place operations like data exploration, various techniques can be done in short amount of time. In contrast to traditional machine learning where data is centralized, it is retained by the participant instead and not shared with anyone else by relying on a decentralized training process. Instead of transferring data to a central entity, as shown in figure 2.8, the training and aggregation process is done on the participants where data is collected. This approach has several advantages over ”traditional” approaches.

2.7.1 Why is Federated Learning needed?

Nowadays mobile devices have the ability to access huge amount of data. This in return can be used for machine learning models, as a lot of data is required for the machine learning models. Speech recognition and image models are some examples. However even in the presence of availability of data, more often than not, they are sensitive. Thus sharing those valuable training data for machine learning models becomes a problem. With time, the amount of data produced is becoming larger and larger. This is where FL comes in handy. To summarize the situation:

1. Data privacy is violated by transferring data. In federated learning the data remains on the participants devices and only the model updates which are received after local aggregation, are sent to the server. This preserves data privacy issues. Data can be used without directly accessing it.

2. Another challenge with data transfer are the data volumes. Raw data can get up to higher numbers really quickly which is why it is not always possible in the first

place to transfer data. FL allows the use of mentioned data without having to transfer them. Furthermore depending on the data volume a data center is needed for handling of that data, as no single device could handle such huge amount of data.

Most machine learning models assume IID data (Independently and Identically Distributed). This means that the assumption that all the data collected represent the whole population, which makes sense because data was gathered from all over the place. With enough data this assumption becomes more realistic. In federated learning however this is not the case anymore. This is because of its distributed nature of the data. Because the training and aggregation are done on local data, the amount could never be this large as the whole population together, thus this needs to be remembered of. By not accessing raw data directly, however there are other problems in federated learning. Communication cost and reliability are major limiting factors here.

2.7.2 Fedaveraging

So before the start of the training process in FL the selection of eligible participants has to be done first, because minimizing the negative impact of the participants they could introduce is a priority. So in short each participant receives a global model or rather a copy of it. Then they start the training on their locally gathered data. After the training, only their update parameters from their models are sent to a central server. No transfer of the raw data is happening here. After the central server has received enough (threshold is flexible) updates it will start the aggregation process in which it uses the update parameters received to compute a new global model. The process is repeated until the goal has been reached (e.g. certain accuracy has been reached). Algorithm 2 shows a possible approach.

The fedaveraging (fedavg) algorithm is divided by 2 segments. One for the server and one for the clients (participants). [15] At the beginning of each round the server selects participants, based on criteria, for aggregation. The criteria for selection, in case of an iPhone as a local model for example, could be, that it is fully charged at the time of selection and that a certain version of the IOS has to be installed on the iPhone. Those selected participants receive the current global model for their training. Then the respective participants start their training by using various already discussed methods such as gradient descent, to produce new weights for their local model. After their training has been finished for this round, they sent their model parameters to the central server back. The server will then aggregate all the updates, to compute a new global model by averaging the update parameters as the name suggests. With the new model, a new round begins and the server will select the participants again.

Algorithm 2: Federated Averaging from [15]

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$m_t \leftarrow \sum_{k \in S_t} n_k$
$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client* k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

2.8 Centralized and Decentralized Federated Learning

FL is all about sharing update parameters instead of data directly. In federated learning there exist multiple ways to achieve this. CFL as well as DFL are core settings in this thesis. These settings differ from each other on a structure level mostly. They are still federated learning approaches so they inherit the ”no data” sharing property and all other properties too.

2.8.1 Centralized Federated Learning

In CFL the architecture looks similar to ”traditional machine learning” architecture where a central server coordinates the training process. But only the update parameters are being shared with the server. At the beginning, the central server will select participant which it considers trustworthy or eligible. Then the chosen participants will receive the initial global model from the central server to begin their training. The participants start their training process on their local devices with their local data which is gathered by themselves. For example, an iPhone can be such a devise as it passively collects data from apps usage and if allowed some other useful data like sleep time and walking distance. After the data gathering has been finished, they move on to the training phase. Finally after the training epochs have been finished, they only sent their update parameters(new model weights) to the central server. The server will collect the update parameters from the participants and aggregate them by using the respective aggregation function(e.g. fedavg) to compute a new global model for this round. A new round starts and the

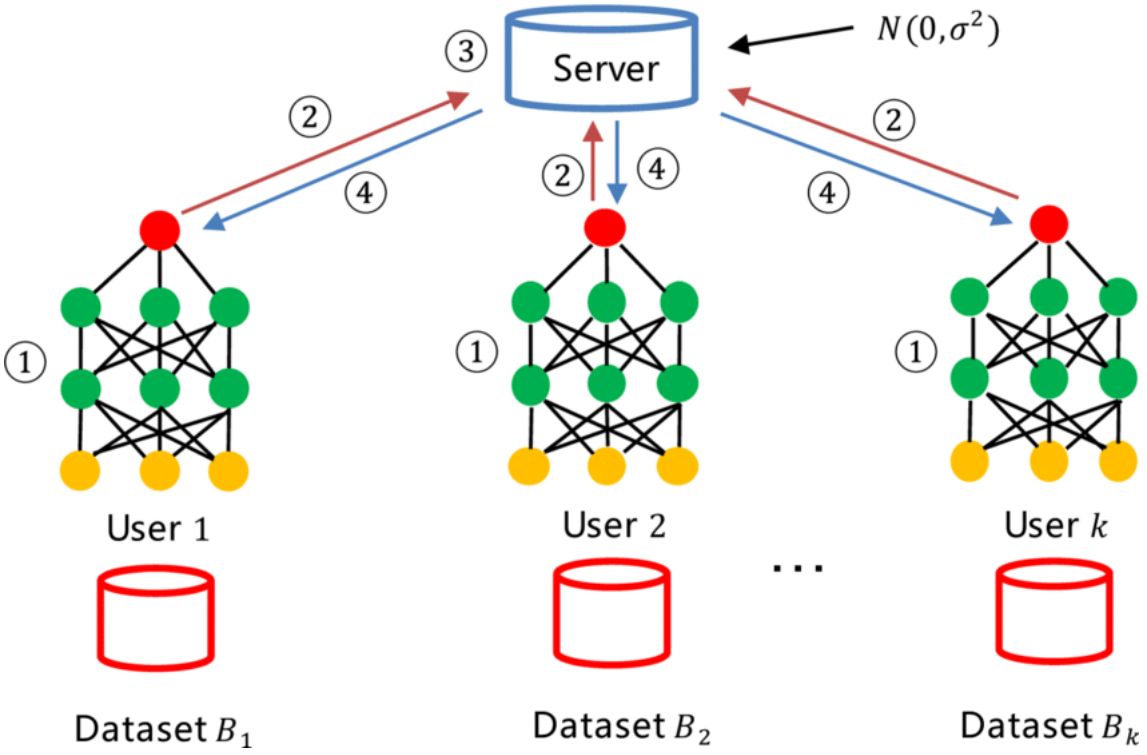

Figure 2.9: A structure similar to the "traditional machine learning" setting

process is repeated. The server will select participants again and the whole process begins anew.

In figure 2.9 the system works as follow:

1. This is where the training process happens
2. Here the participants share their update parameters with the server
3. In this step the aggregation is being done
4. At last but not least, the redistribution of the new global model takes place

Because a central entity is responsible for selection of participants and the aggregation, it enhances coordination and synchronization of the whole system. The complexity is also manageable as the whole process can be managed from a single point. But having a point where everything is located brings disadvantages with it. For example, if some decides to attack the system, they only have to consider one point to attack as the central server is the most important location in the whole structure. Furthermore since all the selected participants have to send their updates to a single entity, the server can also become a bottleneck of the whole system. In case of an fatal error of the server, even the whole system could be shut down. This is called a single point of failure.

2.8.2 Decentralized Federated Learning

DFL works as the name suggest in a decentralized manner and this not only on a data related level but on the nodes and communicating method as well. In DFL there is no central server anymore which was responsible for basically most of the process. Instead now the participants overtake the role of the server. [16] There are settings where every single node could take the role of a server. What does that mean and how does it exactly work? Figure 2.10 shows an example of a fully connected DFL structure.

There is a network of participants. A participant is a person with a mobile device which can gather data can also conduct analysis and training on the data set. Furthermore it is able to share the update parameters. The question now is, where do the participants sent their update parameters to, after they finished their training.

If there is no central sever anymore and anyone can act as a server then the will sent them to their neighbors. A neighbor is a node which has an edge to the current node. In this example this could be the nearest participant to the current participant. They will exchange their update parameters with each other. In DFL there are ”aggregators” and ”trainers”. The trainers only share their update parameters with their neighbors as well as receiving the update parameters but they do not aggregate the updates. This means that they do not compute a new model after each round. They only exist to perform training on the data set to generate update parameters. In this example it can be compared with a weekly report of the mobile device as the ”training on the data set”. The trainer will sent the report to our closest participants but will do nothing further. The aggregators are the ones that do the aggregation. Every aggregator will perform aggregation after it has received enough update parameters from its neighbors. They also generate a new model each round. In regards to this example, it could be a aggregation of all the weekly reports they have received. With the new found model, they for example, know now on which aspects of a mobile device they have to focus more on so that the next weekly reports are going to be different from the last ones containing more needed information.

If all the aggregators reach consensus after set amount of rounds, then the training process will be considered as finished. There is no more one entity responsible for the majority of the process but instead there are more entities now. This directly solves the single point of failure problem which is prevalent in the centralized federated learning setting.

But this setting comes with some downsides as well. First, because every node has to share its update parameters with ALL of its neighbors, there will be a lot of communication overhead. This means that the whole process takes longer. Second maintaining control is more difficult critical information is not stored in one place anymore. In CFL everything is stored at the central server, the search is easy. In DFL this is no longer the case.

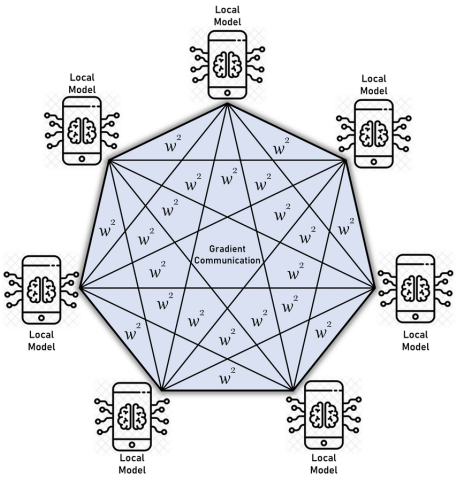

Figure 2.10: Example of a fully connected DFL structure

2.9 Byzantine Fault and Robustness

Because of its distributed nature FL is vulnerable to attacks and errors. The most prevalent one is a so called byzantine fault. [17] A byzantine fault is a condition of system in which parts of the system may fail or not work as intended anymore and there is insufficient information on, if a part has failed or not. A byzantine fault it is difficult to detect because a definition of what is ”wrong” has to be defined first. Behaving differently does not necessary have to be ”wrong”.

FL does not have the property of independent and identically distributed (i.i.d.) data because each local device conducts its training on its own data which does not have to be the same as another local device, in fact that is rarely the case. Thus, every participant will behave differently based on the data they get.

The origin of the name comes from a generals problem. [18] Considering there is a number of generals having to consider attacking a castle. There are two options. Either they all attack together or they do not attack and retreat. If they act on split decisions then it would end up much worse than the two other options because they would for sure suffer more losses, by ensuring chaos. Thus, they have to vote in order to make a decision. But they are restricted by their location, as they cannot see each other so they must rely on messengers to deliver their messages. Figure 2.11 shows an example of the generals problem with six generals.

The problem is more clearly visible when there are traitors among the generals. For example 11 generals, five of them are voting for an attack and five of them are voting for retreat, there could potentially be one general which is trying to sabotage the army by sending one vote for attack to all the attacking ones and one vote for retreat to the retreating ones. This would exactly lead to the worst possible outcome.

The general would be nodes in this case and sending messages are the connections between the nodes. A system which can still function without a major impact or decrease in performance in presence of byzantine faults is called byzantine tolerant or robust system.

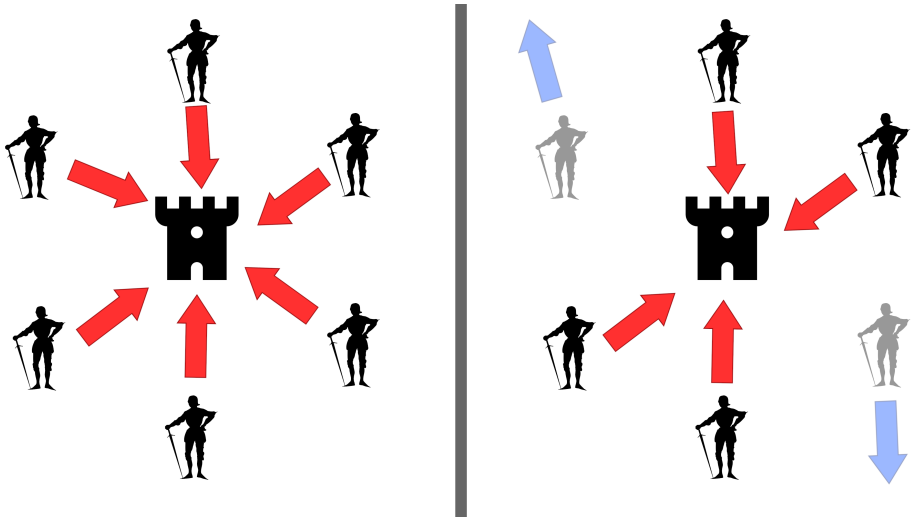

Figure 2.11: Left: Initial positions, Right: 2 Generals are voting for retreat and 4 are voting for an attack

This is an attribute especially important for FL because some participant could have malicious intends. Furthermore it is easy for an attacker to create a fake clients, to purposely infiltrate the system. In order to achieve byzantine robustness observers or mechanism that prevent the malicious participants from going on a rampage are needed.

Coming back to the generals problem byzantine robustness can be achieved by having a majority agreement of the non traitorous generals. In case of missing messages there should be a special value for vote, for example the null value. If the majority of the votes are null values then there should be a pre designed procedure for this case. It could something like retreat, because missing votes means that they are probably not fully prepared for an attack.

Chapter 3

Related Work

In this chapter insight into most works related to byzantine robust aggregation and which type of attacks exist in FL is given. With this knowledge, potential weaknesses can be detected in order to develop a new algorithm which aims to fill the gap.

3.1 Attacks in FL

Why are aggregation rules needed? Obviously from time to time some models could be malfunctioning or the data provided is corrupted. But the main reason is that there are adversaries that try to poison the FL system to gain advantages [19] from it like data or installing a backdoor. In this section some of the attacks will be explained.

3.1.1 Data Poisoning

Data poisoning are attacks where the adversary attempts to manipulate the data. If a model uses said manipulated data its result will be influenced as well, because it is training on manipulated data. Most common data used by machine learning model are labeled, meaning they have a informative label added to the data so that a machine learning model can learn from it. A picture of an elephant with the label "elephant" is an example for labeled data.

To manipulate such data, flipping the labels is an easy way to do so because no assumption of loss function or deep neural network structure have to be made. It is very time efficient and an popular option especially in federated learning as it is often executed on devices that are not very well protected such as phones, local computers etc. Flipping the label means changing it[20]. Out 50 labelled pictures of animals, some of them could be changed and the model would suffer in accuracy already. There are several research on effectiveness on some attacks and accuracy model out there.

The animal in the middle of the figure 3.1 is clearly not a dog. But due to label flipping the machine learning model will use this as a guideline and always identify a bird as a

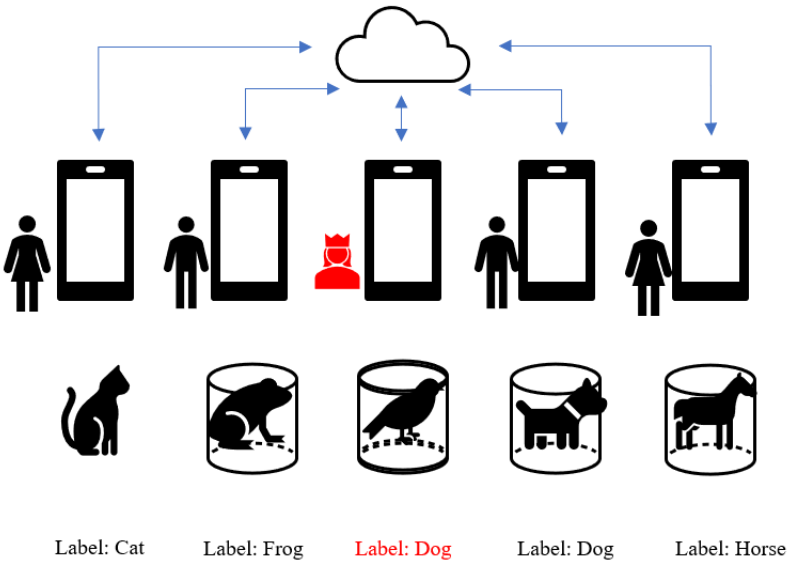

Figure 3.1: The middle participant's data has been poisoned

dog with this data pair. If a person has to learn new things and receive lectures about the subject but with some of the information being incorrect, it is hard for them to identify which one are the incorrect ones, as he has little knowledge about this subject. This is the same scenario the machine learning model goes though. It has no knowledge at the beginning of the training, and if the label is already wrong, then it will only continue to learn wrong things making it worse and worse.

3.1.2 Model Poisoning

Apart from data poisoning there exist model poisoning as well, where the attack does aim for the model parameters itself [21]. Normally in federated learning the training is conducted on the local devices. By manipulation of the model parameter the model will be affected and it will differ from the other models in the network. Figure 3.2 shows an example of a neural network and a neural network with poisoned neurons. So the poisoned model will always share manipulated update parameters. The neural network is involved in the training process so knowledge of the architecture is required to manipulate the training process. [22] Sneaking in a fake gradient is also a possibility. The impact of fake clients on a FL systems have been researched by Yann Fraboni et al. [21] They found out that many aggregation methods rely on outlier detection such as median or trimmed mean. Malicious attackers providing abnormal updates would go unnoticed here.

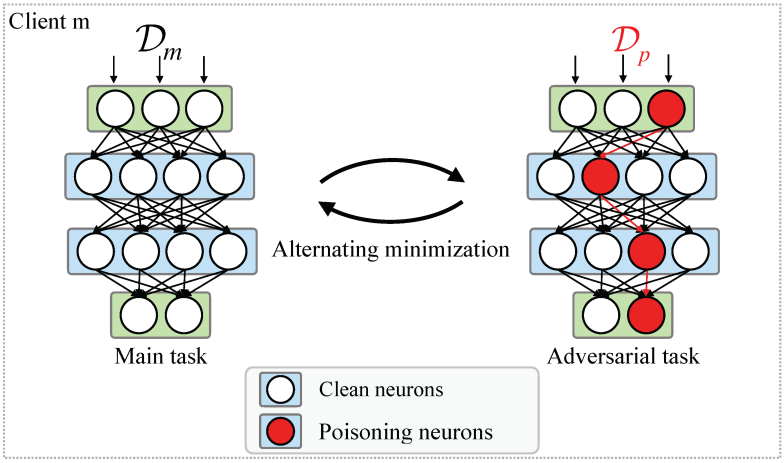

Figure 3.2: On the left side, everything works as intended. On the right side however, some of the model parameters(neurons here) have been changed and the output of the training round will be different than it would without manipulation

3.2 Aggregation

Aggregation is a process to basically compile a typically large amount of data into something more accessible In terms of machine learning, the central server in CFL or the aggregating nodes in DFL receive a lot of update parameters. What the aggregator does is combining the information it has received to generate a new output which then can be used. The goal and output here is a new global model. So in FL, the aggregator generates a new global model for further training process rounds. Fedavg is one of the aggregation rule that has been mentioned before. This aggregation rule simply takes the averages over all the update parameters. With that, it will calculate the new weights of the model and shares it with the relevant participants.

Fedavg is not the only aggregation rule that exist. When talking about aggregation it is necessary to filter out potentially malicious clients (participants). Not every update parameter of a participant has the purpose of improving the machine learning model. In case of an attacker they have other goals to realize and work against our machine learning model. In order to protect from attackers and malfunctions byzantine robust aggregation rules are needed. For example if only one client is infected and acts weird, the whole system should not be affected as much. In this following chapter some byzantine robust aggregation rules that have already been used in the past are being explained.

3.3 Byzantine Robust Aggregation Rules

3.3.1 Krum

Krum is an aggregation rule proposed by Peva Blanchard et al[23]. The main idea consist of identifying which nodes to aggregate. One way to do this would be to have some sort of criteria which only the honest node would have. Assuming that each honest node n produce a vector V(t,n) where t is the round and the number of honest nodes outweigh the malicious ones, a malicious node m is going to try to produce a vector (t,m) which is a random vector instead of one it is supposed to produce.

The server then for example computes a vector by applying a deterministic function to the vectors received. The correct nodes are assumed to produce vectors near the "real" vector An update from a malicious node will differ from the computed vector a lot and will not be considered for aggregation next round. The function used to compute such vector is called "krum-function" which will not be explained here. Using the krum aggregation rule, it is applicable to many machine learning models as it is easy to implement and supports scenarios where the honest nodes outweigh the malicious ones.

3.3.2 Trimmed Mean and Median

A more mathematical approach is an aggregation based around statistical methods. Under different conditions of the loss function, certain statistical approaches perform better than others. Mainly 2 approaches haven been suggested by D.Yin et. al. [24] First one being coordinate-wise median and second one coordinate-wise trimmed mean. Without going into details how they actually work, the main idea behind this, is to let the majority carry the weight of the contribution. Because the mean is more vulnerable to outliers an adjusted version of it has been proposed. Before taking the mean of the contribution a certain percentage of the data is being cut off from both sides. With this it can be ensured that the extreme outliers do not impact the result as much as before. While the theory behind this is rather difficult the actual implementation is not. Using only statistical methods such as mean and median the implementation looks straight forward and is used in many machine learning models. Furthermore the algorithm is time efficient.

3.3.3 FLTrust

Moving on to the more advanced algorithm "FLTrust" (FLT). FLT works on a reputation system. Assume that every node is equally trustworthy at the beginning. With time (rounds/epochs) their reputation can and will increase and decrease based on their performance in relation to the average. Generally speaking if the participant performs much worse than the average they lose reputation. Otherwise they gain reputation. This thesis will not elaborate on how exactly the reputation is going to be computed. This can be seen in this paper [25]. The reputation of the nodes affect their contribution weight. The better the reputation of a participant, the "more important" its contribution becomes.

Vise versa if its reputation is very low, its contribution does not matter as much. Such a system is very adaptable, as the contribution changes overtime based on the performances of the respective participants. The thing to be concerned about most is the fact that it needs a warm up in the sense of reputation, because in the early rounds there is no reputation set yet.

3.3.4 Bulyan Aggregation

Bulyan aggregation rule aims to enhance the already existing ones like Krum and Median. (El Mahdi et al)[26] show that in in case of non-convex loss function and higher dimensional neural network, convergence is not enough. Thus a stricter rule is needed to achieve byzantine resilient results. In the aggregation step the aggregator has to choose which updates in will take for the aggregation. Updates received from participants come in form of vectors, as the Stochastic gradient descent gives us a vector. For higher dimensional neural networks if the model follow the already known aggregation rules it may end up with with vectors with one of its coordinate having extreme values. This is why in addition to being byzantine resilient in sense of convergence, Bulyan also makes sure that each coordinate of the proposed vectors is also agreed on by a majority of vectors similar to a voting system. This rule can be applied to any existing aggregation rule. To ensure mentioned properties Bulyan does following. Let A be a byzantine resilient aggregation rule. Now let A choose from the received updates (vectors) closest to A. With Krum for example this would be exactly A. Afterwards remove the chosen vector and add it to a selection set S. Now repeat the process of choosing and removing again as long as the cardinality of the selection set S is smaller than a condition. With this it is ensured that S contains a majority of non byzantine gradients.

3.3.5 Scaffold

The disadvantage of using FL is non IID data. The results are slow convergence of the global model which leads to longer training times and even inaccurate results. [27] It introduces a so called ”drift” in the update parameters of the nodes which causes the slow and unstable convergence. Scaffold is a algorithm which tries solving this drift problem. Intuitively the idea is as following. The algorithm estimates the update directions for both the server and the participants. The difference is the estimated ”client-drift” which is used to correct the local models. With this correction, the convergence happens much earlier thus reducing communication cost and other resources. This means however that the server has to maintain additionally to the server model, a separate parameter for each participant.

3.4 Federated Learning Ensemble Techniques

In machine learning combining several models to obtain better performance than a single model could do, are called ensemble techniques. Ensemble techniques benefit from the collected knowledge from all the involved models. Essentially ensemble techniques aim to get results ”better” than the sum of its parts. The philosophy behind the idea is that a single model cannot cover everything that is involved with machine learning processes such as security, training techniques and so on. And where one model fails to do it, another model succeeds. By combining the knowledge of all the models together, the the ”ultimate” model can be achieved, or at least that is the idea. In federated learning, where

training is done locally and data distributed across multiple nodes, ensemble techniques can be used to strengthen robustness and boost accuracy.

3.4.1 Fed-Ensemble

This technique is based on averaging over multiple models [28]. In a normal FL setup there exist one server which aggregates updates from participant. Fed-Ensemble aims to have multiple aggregators. They are independent from each other and each round fed-ensemble assigns one of them to selected participants to train on their local data. Eventually they all learn from the entire data set allow improvement of performance by averaging over all of their predictions. Such process allows for more accurate predictions because multiple models are involved. It also is more robust to attacks as poisoning only a single model will not be having enough impact to affect the whole system.

3.4.2 Knowledge Distillation

There is more ways to utilise multiple models than just averaging over them. More often than not it is computationally expensive to have multiple models training on data. This is where knowledge distillation comes into play. Knowledge distillation makes use of a student teacher system. The teacher model is trained on a large amount of data which in return will have good generalization capabilities and knowledge[29]. Such model is generally not used for deployment as it is cumbersome and requires huge computation.

Once this model has been trained, a different kind of training can be used for smaller model more suitable for practical use called distillation. The goal is to transfer the knowledge from the ”big model” to the smaller ones. Knowledge is transferred by creating ”soft targets”. These ”soft targets” are probabilities produced by the teacher model. They serve as guidepost or hints for the student models. By mimicking the teacher model’s output distribution, the participant models can learn from its knowledge.

3.4.3 Federated Multi Task Learning

With more and more devices storing massive amount of data and getting more computational power, federated learning has also increasingly becoming popular. Training models on local devices are more attractive than ever because of the problems with ”traditional machine learning”. However the limitations of federated learning are not to be underestimated such as high communication cost, because of the relatively high number of nodes in a network. Thus, computational power and communication capacities can be limited depending on the resources available. Additionally the fact that there is non IID data present, the data amount and type can also vary from node to node. [30] As of now, the aim has always been to train one model across the network. This method however aims to train multiple models at once. The goal is to fit multiple separate but related models with a multi task learning framework. [30]As with the huge number of nodes the aim is to having some of the nodes train on additional models.

In this table the byzantine robust approaches are being put into comparison mainly in terms of resource management.

Byzantine Robust Aggregation Methods					
Reference	Year	Approach	Type	Communication	Complexity
[23]	2017	Krum	Aggregation	Little	scalable
[24]	2021	Trimmed Mean	Aggregation	Little	scalable
[25]	2021	FLTrust	Aggregation	Medium	scalable
[26]	2018	Bulyan	Aggregation	Little	scalable
[27]	2019	Scaffold	Ensemble	Large	cost heavy
[28]	2021	Fed-Ensemble	Ensemble	Small	medium
[29]	2015	Knowledge Distillation	Ensemble	Medium	scalable
[30]	2017	Mocha	Ensemble	Small	scalable

Communication refers to the amount of communication between the nodes. In CFL the main point of communication are the central servers with the participants. In DFL the communication between the participants themselves are the main focus. Complexity refers to how well the system can perform with increasingly number of participants. Some algorithm work well in small scale settings but with big number of participants, they start to perform worse. Most of the are designed though, to function well in large scale settings as well.

3.4.4 Which Technique is the best one?

There exist many more aggregation rules which are much more complex than the one presented, and the question arises, which one is the best one. Our goal is to maximize protection as well as minimize the resource consumption. These two words are already good indicators for the selection of an aggregation rule. In many cases a trade-off between protection and resource consumption is present. In theory a function can be designed, which maximizes the protection amount while minimizing the resource consumption amount. This alone could prove to be difficult already, as the ”protection amount” is not a clear number. It is probably a threshold based on the machine learning model as well as the setting it is in. Setting is the next keyword which is important. Some aggregation rule are not as scale-able as others. Take for example fedavg. Averaging over all the update parameters can be done in linear time. But if there is an aggregation rule which does not work with linear complexity, then the time it would take to do the computation would drastically increase with each additional node in the topology.

Chapter 4

Design and Implementation

4.1 Motivation

Now that criteria or area, security and robustness that can be improved on have been explained, this work will propose an algorithm which can enhance and strengthen the robustness of an aggregation rule. As DFL has not gotten much attention as its counterpart CFL so are the byzantine robust aggregation methods[31], [32]. As such this thesis try to enhance the most popular existing aggregation such as fedavg, krum, median and trimmed mean. As they are relying on some majority of participant to dominate the aggregation the number of participants are important.

With less participants it is easier to infect the system and the variance is bigger. A network of 200 nodes is much more stable than a network with 20 nodes as the infection of one node does not have a large impact. This may not be a important factor in CFL where the central server takes all participant into consideration. In DFL however the situation looks different. The aggregators in a DFL only take their neighbors into consideration for parameter sharing. This can sometimes lead to a situation where a node does not have enough neighbors for a reasonable aggregation, as the question arises, how can a node trust its neighbor if it only has itself as the reference.

4.2 Neighbors-Problem

Considering a DFL system as a graph there are constellation where nodes only have one neighbor. Other problematic constellation are when there are more central located nodes with many single connections. In such case if the central node is malicious then it is easy to infect all the edge nodes because their only connection is to the malicious one. Furthermore because of the neighbor nature it is possible that local areas are being poisoned.

For a node standing on the edge between a corrupted area and a clean one there are two options. If the number of malicious neighbors overweight the honest ones then it will

be poisoned as well as there are too many malicious updates for aggregation. Otherwise the accuracy will be decreased stronger the higher the relation of malicious nodes is in contrast to the honest ones. In either case it would impact the performance of the node negatively.

The goal of this aggregation is to have an optimal selection for aggregation in case it is not provided naturally. To do so there are three iteration where with each iteration the goal is to make the system more robust.

In the first step this thesis will solve the one neighbor problem. In case of not having enough natural neighbors, adding more neighbors will be done first. This can be done by adding the neighbors of its neighbor. By doing so it is ensured that there are at least three or more nodes for aggregation. With three or more it is also ensured that there is a majority of nodes (positive and negative). With the assumption that not more than 50 percent of are malicious, on average there will be a majority of honest nodes for each node in the network.

What remains is the fact that if the neighbors of the neighbor are also poisoned (local poisoning) it would lead to the same problem. Also some topology do not allow this ”neighbor of neighbor” selection.

In the second step these issues have to be addressed. First an algorithm better than ”neighbors of your neighbor” has to be designed. After that the optimal number of nodes for selection need to be calculated. With a network with n nodes and m malicious ones and $m < n/2$, if a honest node has been selected, then the total number of available nodes for the second selection is decreased by one and more important, so does the number of honest nodes, because a honest node just got selected.

In the initial state the number of honest nodes over weigh the malicious ones. This is why with each node further after the first one, the the chance of having a successful aggregation where successful means that the model aggregates with more honest nodes than malicious ones, increases. This continues as long as there are still more honest nodes than malicious for selection. As mention before though, the relation of honest to malicious decreases with each selection and at some point they are equal in number. Further selection beyond this point would not benefit us in any ways as the have equal chance of selection a honest or malicious one and beyond this the chance of a malicious on is greater than a honest one.

An algorithm that would perform better in terms of selection the neighbors would be ”neighbors of at least k-distance” where k is a natural number. So for a network like this, neighbors of distance of at least three would be good to avoid said problems. The distance must be adjusted depending on the topology of the network.

For he ideal amount of number for selection a network with n nodes where as m are malicious is being considered. After picking a node, the total amount of nodes decreases by one. It is beneficial for us to pick another node as long as there are less malicious nodes than honest nodes as the probability of picking a honest node in this case is higher. The selection stops when the remaining number of malicious nodes are equal to the honest one.

4.3 Neighbor-Selection

Algorithm 3 shows a possible solution for the neighbors problem. Neighbor Selection simply goes over every node in the network and add neighbors of any certain distance until the required amount has been reached, as shown in the Algorithm below. This ensures that there is the right amount of neighbors for every node. If there are no nodes of k-distance anymore to select and the required amount of nodes are not satisfied yet, then random nodes in the network which have not been selected yet, are going to be selected until the required amount is reached.

Algorithm 3: Neighbor Selection

```
Neighbor Selection:

For each node in graph do:
    For each neighbour in adjacency_list_of_node do:
        if len(neighbor_list) < threshold:
            Repeat
                add nodes of k-distance to neighbourlist
            until len(neighbor_list) == threshold
        Endif
    Endfor
Endfor
```

4.4 Neighbor-Selection with Reputation

To further strengthen the aggregation a reputation system can be introduced, in which every node keeps track of its neighbors contribution. Algorithm 4 shows an approach including the reputation parameters.

Similar to FLTrust where the server keeps track of the participants, in DFL everyone can act as an aggregator. The nodes set the reputation for each neighbor at the beginning to neutral. With each aggregation they increase or decrease the reputation of their respective neighbor based in its performance in relation to the other neighbors. Just like FLTust if they perform worse than the average (consisting of neighbors of the node's perspective) the aggregating node will decrease the respective reputation. Reputation impacts the weight of the contribution. If a neighbor of node has a bad reputation, then its contribution will not be weighted as much as a contribution from a neighbor with good contribution. With a system like this the nodes can adapt dynamically and even consider choosing new neighbors if the reputation drops below a certain threshold.

Algorithm 4: Design with reputation

```
Neighbor-Selection with reputation:

For each node in graph do:
    For each neighbour in adjacency_list_of_node do:
        if reputation > reputation_threshold:
            add neighbour to neighbourlist
        Endif
    Endfor
    if len(neighbor_list) < threshold do:
        Repeat
            add nodes of k-distance to neighbourlist
        until len(neighbor_list) == threshold do:
    Endif

    after aggregation do:
        for each neighbor in neighbourlist do:
            calculate new reputation_score
        Endfor
Endfor
```

4.5 The Algorithm

This chapter provides insights into the implementation based on the algorithms discussed in the chapter before. First the selection of neighbors as well as the transformation of the graph is discussed. Second the framework on which this implementation is done on is going to be explained. Afterwards the implementation on the framework is explained, and last the limitation of this framework is discussed. Note that everything has been implemented using python and its respective libraries.

4.6 Graph Discovery

In a real world scenario there is a given random topology without the knowledge of the topology. This could be a scenario where the participants phones are being used to conduct training on collected data and share their updates with their geographic neighbors. Such, the exact number of participants and neighbors are unknown. Because of this, a discovery of the neighborhood or even the whole network has to be done first before proceeding further. From a more theoretical standpoint a graph represents the network and the nodes represent the participants. Therefore a graph discovery has to be made first. There are many graph traversal algorithm out there already. A modified version of Depth-first-search (DFS) is being used, where the number of visited nodes are also being kept track of during the process, as in a real world scenario it is not that simple to ”visit” another

node. In order to to that it is necessary to establish a connection to that node with a protocol. Because of the unknown structure, a counter has to be present to count the steps.

As the implementation of the graph discovery is not the main focus and only done theoretically the computation cost with some assumptions will be included in later chapters. In the chapter evaluation an example will be provided on how the cost changes with the size of the network.

4.7 Modification of the Graph

This step of the algorithm requires us to modify the graph such that the conditions are satisfied. The conditions were that if a node has only one neighbor or not enough neighbors, depending on the threshold, then it has to add more neighbors according to the searching algorithm. To achieve that iterate over every node in the graph and check if they have enough neighbors. Algorithm 5 shows the iteration over every node. If they have not enough neighbors the algorithm will search for appropriate neighbors and add them until the required amount has been reached. If after the adding neighbors algorithm there are still not enough neighbors, then the algorithm will start adding in random nodes.

Algorithm 5: Modification of the graph using calculating neighbors method:

```
for node in graph:
    if len(graph[node]) < threshold:
        calculating_neighbors(graph, node, distance, threshold)
    if len(graph[node]) < threshold:
        add_random_nodes
```

As in python a graph can be represented using a dictionary. Therefore a key represents a node and its value their list of neighbors. So by taking the length of the list the numbers of neighbors can be determined. If they do not meet the condition, add more neighbors with the method `calculating neighbors`. `calculating neighbors` requires four parameters. The graph itself which has been discovered, the node which needs more neighbors, the distance of neighbors. Distance and threshold are obviously being defined at the beginning. The method calculating neighbors is being explained in more detail in the next section.

4.8 Adding Neighbors

The method calculating neighbors is used to add more neighbors. It works similarly to a graph traversal as the algorithm needs to go to the nodes of k-distance to add them. Breadth-first-search(BFS) visits all the sibling nodes first before the child nodes. Therefore every time it has visited all the siblings it means that it is one distance further away from the original node. A timer is being used to keep track of how many sibling

nodes remain before advancing to the next depth-level. Algorithm 6 shows the design of the depth parameter. By keeping track of the depth as well, it can tell when it has reached the desired distance. A queue is used for the of nodes that are being visited next.

Algorithm 6: Demonstration of keeping track of the depth

```
if time_to_depth_increase != 0:
    time_to_depth_increase -= 1

if time_to_depth_increase == 0:
    depth += 1
    pending_depth_increase = False
```

The depth is initially at zero. In the first round it increases to 1 as it deques the root node and get its neighbors. Based on the neighbors list, it sets the timer to this length as every time it visits a node in this list, it decreases the timer by one. Upon reaching 0, it has visited all the sibling nodes meaning it advances to a deeper level which means that `time_to_depth_increase` has reached 0. Algorithm 7 shows what happens when the parameter has reached 0. The timer is being reset to the length of the queue.

Algorithm 7: Timer reset

```
if not pending_depth_increase:
    time_to_depth_increase = len(queue)
    pending_depth_increase = True
```

Given a simple graph like this:

```
Graph = {
    "0": ["1", "2", "3"],
    "1": ["0"],
    "2": ["0"],
    "3": ["0", "4", "5"],
    "4": ["3"],
    "5": ["3"],
}
```

With the condition that the threshold is half of the nodes and the distance three the nodes "1", "2", "4" and "5" do not have enough neighbors. Therefore the algorithm has to return us a graph where every node has at least three neighbors with the neighbors added having the distance three from the original node. Running the algorithm with the setting above a new graph is being formed:

```
New Graph ={
    '0': ['1', '2', '3'],
    '1': ['0', '4', '5'],
    '2': ['0', '4', '5'],
    '3': ['0', '4', '5'],
    '4': ['3', '1', '2'],
    '5': ['3', '1', '2']
}
```

On the first glance every node has three neighbors which is half the total number of nodes in the graph. Second for 1 and 2 their new neighbors 4 and 5 are indeed 3 distance away from them. The same thing can be said about 4 and 5. Their neighbors 1 and 2 are also of distance 3 away from them.

4.9 Fedstellar Framework

Fedstellar [33] is a modular, adaptable and extensible framework for creating centralized and decentralized architectures using Federated Learning. Also, the framework enables the creation of a standard approach for developing, deploying, and managing federated learning applications. The framework enables developers to create distributed applications that use federated learning algorithms to improve user experience, security, and privacy. It provides features for managing data, managing models, and managing federated learning processes. It also provides a comprehensive set of tools to help developers monitor and analyze the performance of their applications.

Fedstellar consist of a front end and a back end. First, a scenario must be created in the front end after starting the application. A name, description and other configuration such as topology and amount of rounds have to be defined first before the training can start.

After the initial setup the application can be started. First the controller will load the configuration provided and start all participating nodes. Each node is running independent from each other and they train just like DFL on their own data set. When a node finishes its training it will sent its update parameters to all neighbors and enters a waiting state in which it will wait for updates from its neighbors. After it has received them or the neighbors timed out (after a set time) it will continue to aggregation and start the next round. This is repeated until the set number of rounds have passed.

Afterwards several statistics can be viewed such as accuracy, CPU usage etc. As mentioned before first thing to do is increase the amount of neighbors for node with only one or not enough neighbor. This framework requires a predefined topology meaning that an initial network of nodes can be used and apply the algorithm discussed above to it and then enter the new topology.

4.10 Implementation of Graph Transformation

As mentioned before there has to be parts where the code will be change accordingly such that the transformation works. Because of the predefined setting nature, a modification the input parameters of the framework before the main process starts has to be done. For this the app.py file will be changed, which is located in the web server directory. This file is responsible for the setup configuration before the main process. It manages the input web page of the web server. Figure 4.1 shows the interface of the fedstellar framework.

Figure 4.1: Interface of the fedstellar framework

Here the input parameters such as title, data set type, topology can be seen, and in the advanced tab even which aggregation rule is going to be used and so on.

All these input parameters are being managed by the app.py file. The input of the topology is the most important parameter righ now, as it needs to be modified. The input are the nodes on the right side. Edges and nodes can be added there. After the input parameters have been set, they are being sent to the back end. In the back end in the app.py there is following function to retrieve all the data:

```
def fedstellar_scenario_deployment_run():
    if "user" in session.keys():
        # Receive a JSON data with the scenario configuration
        if request.is_json:
            # Stop the running scenario
            stop_scenario()

            data = request.get_json()
            nodes = data['nodes']
            scenario_name = ...

            args = {
                "scenario_name": scenario_name,
                "config": app.config['config_dir'],
                "logs": app.config['log_dir'],
                "n_nodes": data["n_nodes"],
                "matrix": data["matrix"],
                "federation": data["federation"],
                "topology": data["topology"],
                "simulation": data["simulation"],
                "env": None,
                "webserver": True,
                "python": app.config['python_path'],
            }
            ...
```

The request line gets us the data from the front end. Then the data is extracted from the json into an object called "args". Args has many parameters and for us the most important one is the "matrix" one. "Matrix" consist of a so called incidence matrix which is a matrix representation of a graph.

By extending the base screen, the advanced view of the framework is shown which allows input of the attacking type, poison amount and so on as shown in figure 4.2

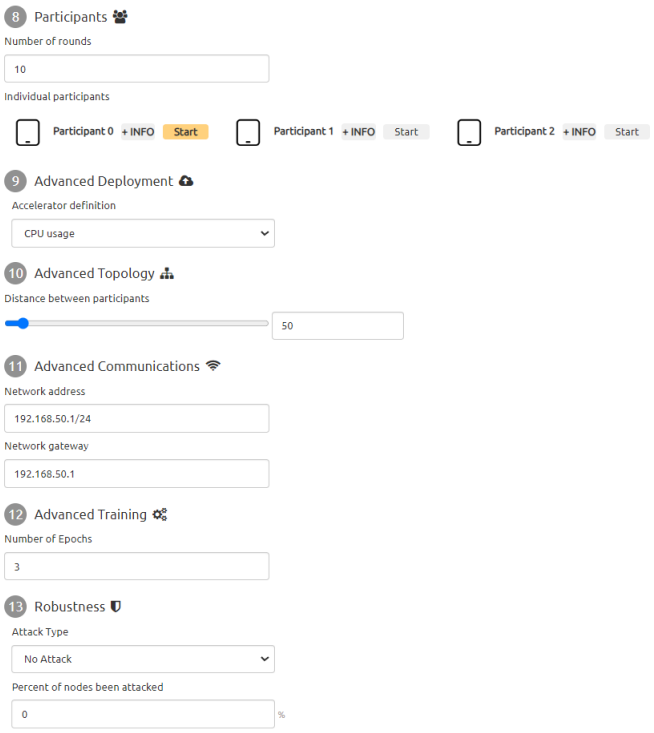

Figure 4.2: Interface of the fedstellar framework

4.11 Incidence Matrix

A incidence matrix is a logical square matrix which consist of ones and zeros for unweighted graphs. The x-Axis and y-Axis are the same, consisting of all the nodes in a graph. If there is an edge between two nodes, then this would be represented by a one in the field in the matrix. For example, the following graph in figure 4.3 does contain an edge between 2 and 3, but not between 0 and 3.

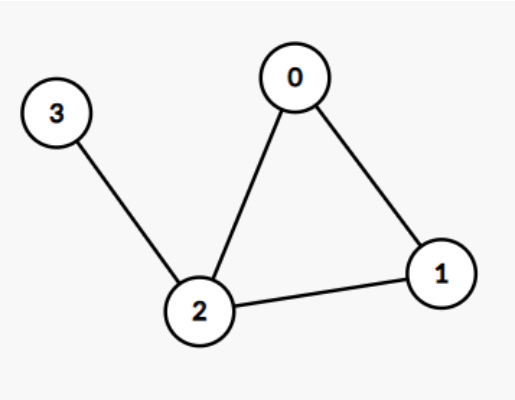

Figure 4.3: A graph with 4 nodes

There fore the matrix contains a 1 in the column and row where the 2 and 3 cross and contains a 0 where 0 and 3 are cross. The corresponding matrix has 4 entries in the x and y axis, so it is a 4 x 4 matrix. Each node is represented in the x and y axis. The corresponding values into the matrix it will look like this:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The python way to represent a matrix is a 2-dimensional array. In case of the matrix above : [[0,1,1,0][1,0,1,0][1,1,0,1][0,0,1,0]] A weighted graph has the value of the weight instead of ones in the matrix.

4.12 Graph Transformation

So after receiving the data from the front end the matrix needs to be changed, such that the condition satisfies the algorithm. To do so consider following. First, every node has to be checked for its condition of neighbors. If they have enough neighbors, then it is okay. Otherwise start adding neighbors to the current node. Figure 4.4 shows the initial graph.

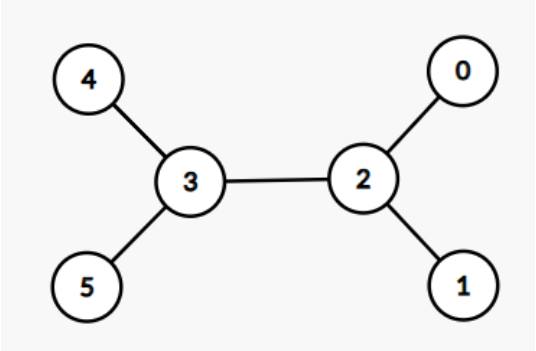

Figure 4.4: This is the graph from the dictionary representation from earlier

Several nodes are missing the required amount of neighbors. So for those add neighbors according to the algorithm, until they meet the conditions. Note that the incidence matrix also changes because mostly edges have been added to simulate the new amount of neighbors. For the whole algorithm four helper functions are being needed.

1. The first function is the neighbor-calculating function. This function takes a node, a graph, a certain distance for neighbor selection and a threshold as input. For the node considered it will add neighbors based on the distance to the current node until the provided threshold is reached

2. The second function is a convert function. It converts the incidence matrix into a dictionary because it is easier handle the graph this way

3. The third function is also a convert function which converts the dictionary back into a incidence matrix because the framework requires such matrix as an input

4. The last function is the one that actually transforms the whole thing. It takes a graph, a threshold and a certain distance again. But this time it iterates over every node in the graph and applies the first function on it. With that every node should get enough neighbors for further processes. If the first function cannot add anymore neighbors due to lack of such nodes because for example, there are not nodes of certain distance to add anymore, then it would just add random nodes from the graph which have not been added yet.

So to summarize, the whole process would look like the following:

1. Get the data from the front end
2. Extract the matrix provided ad the beginning on the web page from the json
3. Convert the matrix into a dictionary for further tasks
4. Transform the graph using the algorithm which adds neighbors for every node
5. Convert the dictionary back into the incidence matrix and plug in the matrix as the new matrix for the training process

After modification of the inputs of the code, following lines are going to be added: Convert the received matrix into a dictionary:

```
dictionary = incidence_matrix_to_dictionary(matrix)
```

Then apply the algorithm to it to find more neighbors if necessary:

```
new_dictionary = number_efficient(dictionary,0.5,3)
```

0.5 is the threshold and 3 is the distance. These are variables that can be changed depending on the needs and situation. After receiving a new graph with the algorithm, it is converted back:

```
new_matrix = convert_dictionary_to_matrix(new_dictionary)
```

Now the the matrix will replace the old one in the args object, in order to save the new topology.

Afterwards the training will start as per usual but with the new matrix.

4.13 Two Examples of a Graph Transformation

4.13.1 Ring Topology

A ring topology is where every node has two neighbors and form a ring structure. In a ring formation, the problem of having only one neighbor will never occur, but instead the threshold of required nodes is the main problem. If one node does not meet the requirement, then all of them do not meet the requirement, as the all have the same number of neighbors at the beginning. So logically the algorithm will continue to add edges until the threshold is met.

Let us consider following graph in figure 4.5

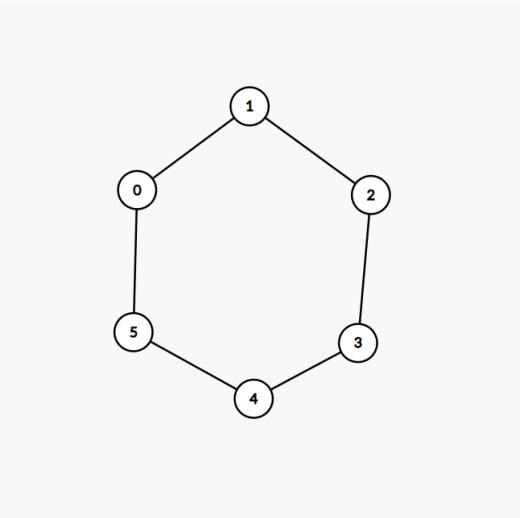

Figure 4.5: Initial Graph

The threshold is half of the nodes (3) and the distance of adding nodes is two. In the first step the node 0 is the start point. This has no special reason and every other node could be the starting node. From here the nodes with distance two to 0 are node 2 and node 4. The node 0 already has two neighbors, so it only needs one more. Node 2 will be selected as a new neighbor as shown in figure 4.6. Again, choosing 4 or 2 does not matter.

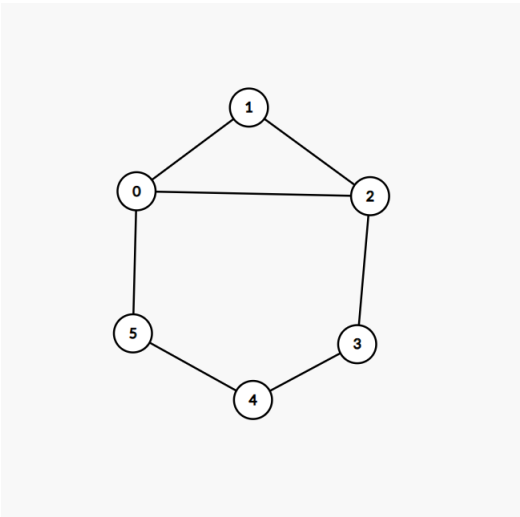

Figure 4.6: Adding the node 2 as a new neighbor to the node 0

Next up is the node 1. It also only has two neighbors at the moment so, the node 3 is being added. The result is shown in figure 4.7.

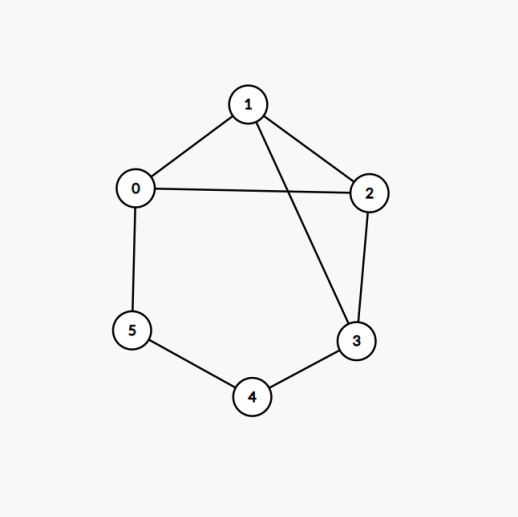

Figure 4.7: Adding the node 3 as a new neighbor to the node 1

The next node is the 2. Note that this node now already has three neighbors, due to the addition from the node 0. So in this case no more nodes are being added The same situation occurs at the next node which is the node 3. This one also has three neighbors, because of the node 1. The next node however, the node 4 has only two neighbors again, so the 0 is being added as a new neighbor as shown in figure 4.8. Note that the node 0 has now four neighbors. This is not a problem, but certainly a optimization problem. There are ways to add nodes in a way such that, only the bare minimum of neighbors are being added into the system, but still satisfy the conditions.

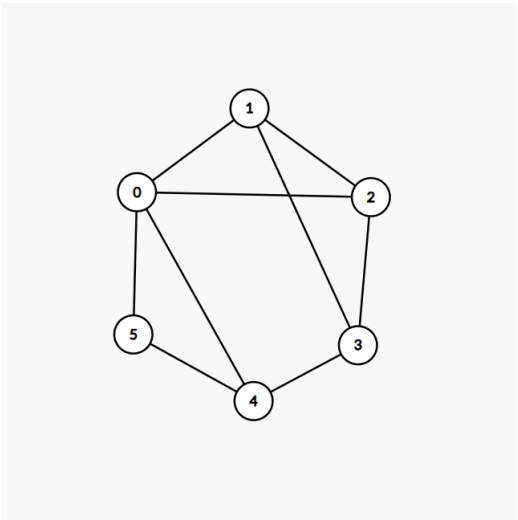

Figure 4.8: Adding the node 0 as a new neighbor to the node 4

Lastly the node 5 adds the node 1 as its new neighbors. With this, every node has at least three neighbors.

4.13.2 Random Topology

In random topology there are no information upfront. So there are no assumptions beforehand.

The initial graph is shown in the figure 4.9.

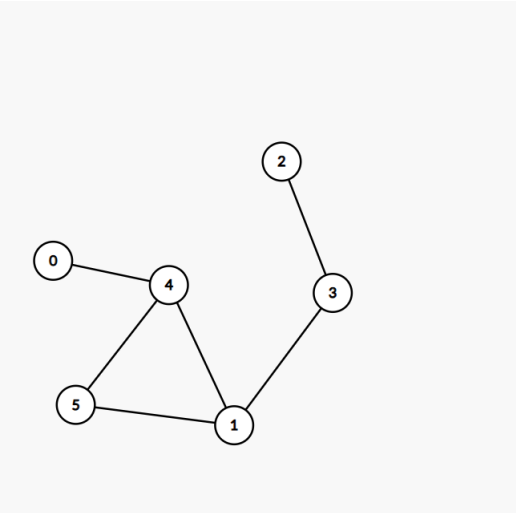

Figure 4.9: Initial Graph

The starting point will be the node 0. This node has one neighbor thus, 5 and 1 are being added as new neighbors, because there are distance of two away, as shown in figure 4.10.

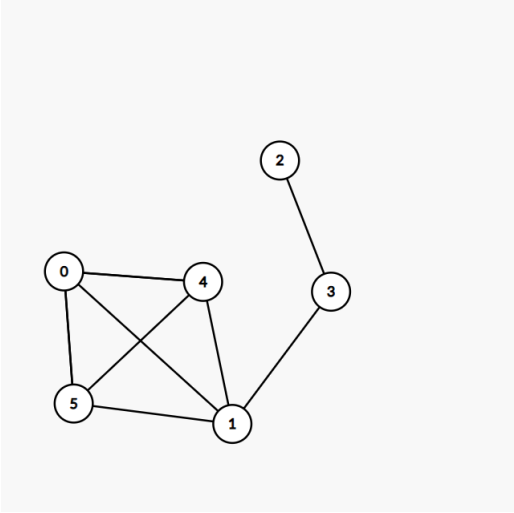

Figure 4.10: Adding in 5 and 1 as new neighbors for the node 0

The node 1 already has three neighbors. The next node is the node 2, which will need two neighbors as well, the 4 and 1. 4 is the random one, because the only node with distance 2, is 1. After that the algorithm will start adding random node in, as shown in figure 4.11.

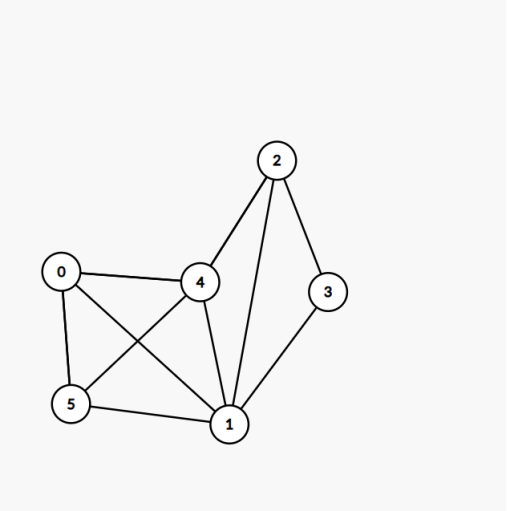

Figure 4.11: Adding in 1 and 4 as new neighbors for the node 2

Finally the node 3 needs one more neighbor which will be the node 4 again. At this point every node has at least three neighbors, as shown in figure 4.12 , so the algorithm stops.

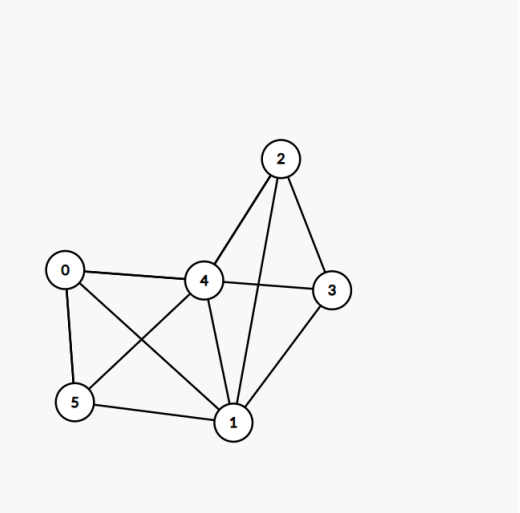

Figure 4.12: Adding in 4 as new neighbors for the node 3

Chapter 5

Evaluation

5.1 Introduction

To evaluate the the performance of the implementation of the algorithm we will have different configurations and compare them in terms of resource consumption like CPU and GPU usage as well as more model related parameters like accuracy. Different scenarios will be created in which each of them the parameters will all be fixed but one and vary the one parameter to see its influence. Also after the single parameter scenarios some of the parameters are also being examined by pair. For example, the combination of topology and poison severity can be one scenario.

The standard setting for the DFL training is as follow:

- It is a fully DFL setting, meaning there are only aggregators
- The data set is always MNIST
- There are 10 rounds with 3 epochs per round

Furthermore following parameters are relevant for the case studies:

1. The topology. The topology is the structure of the graph. The configuration of how it looks like is important here. For example, a fully connected graph will not be affected in any way from the algorithm, as it cannot have more edges than it already has, because it already has the maximum amount of edges. A ring structure instead has much more room for edges. With this variable being flexible the impact of the algorithm on different topologies will be investigated:
2. The number of nodes or the size of the graph. Depending on the size of the graph there is more room for nodes to connect to each other. On average for a random graph there will be more connection. There is also more connection to be made possible so the algorithm has to add more edges in general if the threshold is always at 50 percent. Eventually this could get out of control for very large networks. Following sizes are being considered:

- Small: Around 5-6 nodes
- Medium: This is the standard size which consist of round 10 nodes
- Large: Around 15-20 nodes
- The poison severity. Obviously more poison means less accuracy and performance but the interesting part is how the algorithm is being affected by it, because it keeps a high amount of connection to make sure that even in presence of malicious nodes, it can work "properly". This also tests the robustness to poison. There are also 3 types of severity of the poison:
  - Small infection: Around 5 percent
  - Medium infection: Around 10 percent
  - Large infection: Around 20 percent

Additionally the attack types are data poisoning and model poisoning. The base parameters for all the variables are as follow: The threshold is also set to be 50 percent of the total number of nodes. The distance is 3. This is valid for all settings, unless specified otherwise.

5.1.1 Evaluation metrics

To evaluate the results 2 types of evaluation metrics are being used. The first category contains the resource related metrics. These include the communication between node in bytes, CPU usage in percent, disk usage in percent and RAM usage in percent. The second category contains the performance related metrics. These include accuracy, loss, precision, recall and the F1-Score.

- Accuracy is the the ratio of the correct predictions made over the total number of predictions. Everything is predicted correct, then the accuracy would be 1, because the ratio is the same number over the same number
- Loss is the distance between the true value and the predicted one. For example, if the true value is 5 and the predicted one is 3 then the loss would be 2. The greater the loss the worse the model is in terms of performance
- Precision is the ratio of true positives over true positives plus false positives. As the name suggest, it indicates how precise the model is. The more wrong predictions the model makes, the lower the ratio gets as the number of false positives increases. With no false positives, a precision of 1 (100 percent) is being achieved
- Recall is the ratio of true positives over true positive plus false negative. Recall basically answers the question on how many true positives has the model predicted correctly. So by predicting something into the negative category wrongly, the recall ratio decreases. If every true positive is being identified as true positive, then a recall ratio of 1 is being achieved.

Fedstellar lists performance for each participant it is not really clear case after taking one participant from each scenario because remember that some participants have more connections to others, so it will depend on the choice of participant. The graphics shown in this section are based on the average. They are only shown for easier reading as it is unreadable to show every single participant on the grid.

5.2 The costs and benefit of applying the algorithm

The cost of applying algorithm are put together the following way. The first step is as discussed in the implementation chapter, the graph discovery cost. A simple graph traversal algorithm like Depth First Search can count the cost of exploring the whole graph. After discovering the graph calculate the number of new connections, that have to be made. This is done by the algorithm. Consider the numbers of new connections and add them to the cost before. The last parameter for the cost are the resource consumption during the training. By adding in new connections, the training will also use more resources. Figure 5.1 shows the resource consumption in comparison between the scenario without the algorithm and the scenario with the algorithm.

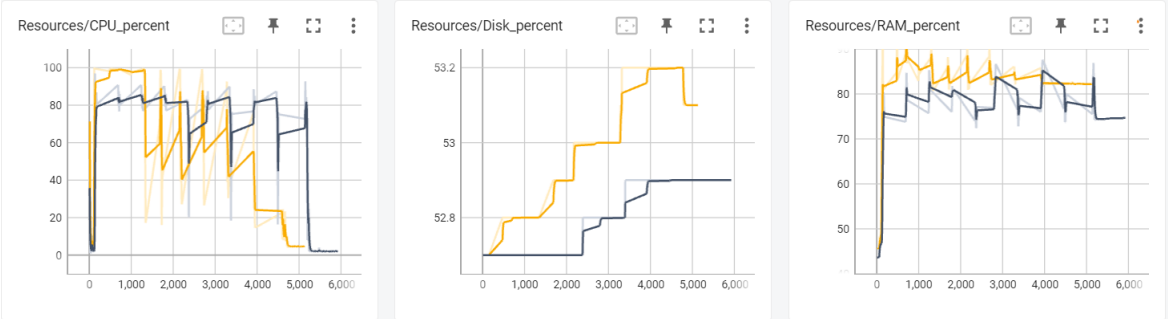

Figure 5.1: While the CPU usage does not show significant differences the disk percent and RAM usage do show a steady increase in resource consumption of the scenario with the algorithm (here in yellow)

CPU consumption on both ends do not differ too much because the framework always tries to fully use the resources it has to finish the training as fast as possible so most of the time, every scenario has max CPU consumption and only decrease as time goes on. Note that because of a strong computer, this parameter may not be affected as much as a computer with less computation power. For disk and RAM percentages, the scenario with the algorithm uses more in most cases. Due to more connections overall, the resource consumption will also be higher.

The benefit from applying the algorithm are boost in accuracy. This is true for all sizes of network. The figure 5.2 shows the accuracy and loss graphs.

The range of the difference is between 5 and 10 percent which is a substantial increase in performance so overall the trade is worth it. For smaller scale runs the increase in resource consumption does not impact us as much. In fact the run-time of both of the scenarios are almost the same. The reason for this could be a very good computer which can handle

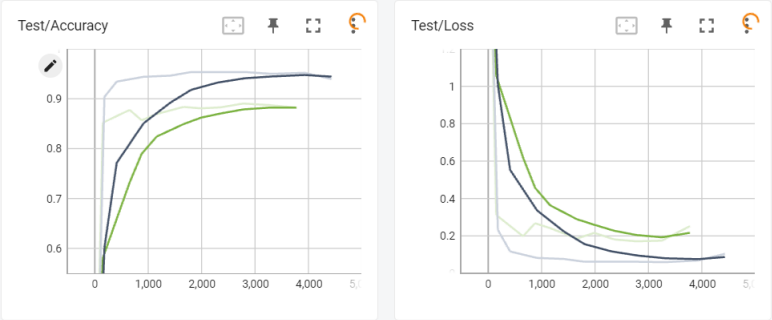

Figure 5.2: Blue: With algorithm, Green: Without algorithm

small scenarios. The precision and recall behave in similar pattern as the accuracy and loss as shown in figure 5.3

Figure 5.3: Blue: With algorithm, Green: Without algorithm

5.3 Case Study 1: Topology

This setting dedicates its focus towards the topology. Three different topologies are being evaluated at the moment which are, the ring, star, and a random topology. In a ring topology, every node has 2 neighbors and they form a ring formation. A star topology consist of one central node and everyone else has a connection to that node as shown in figure 5.4. They all have ten nodes. The poison amount is small (5 percent). The attacking type is data poisoning.

First of all in a ring formation, the number of initial connections is also 10. So the maximum amount of connections to be added is 30, because every node needs 3 more neighbors to reach the threshold of 5. Now this is the worst case scenario. The algorithm is not optimized for selecting the optimal solution. For example, if it has to choose between 2 nodes to add, it will choose random. Also if there are no more nodes of the set distance to add, then it will add random nodes anyways, so the total number of additional connections vary. For example, the average new connections added in a ring topology with 10 nodes, is around 16. In comparison to a fully connected graph with 35 new connections, a lot of resources is being saved here.

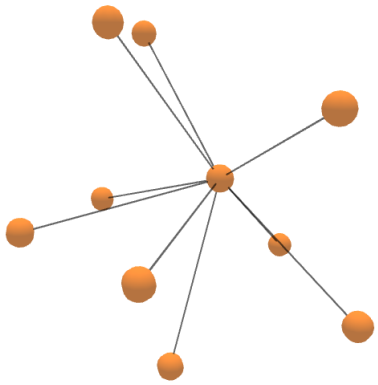

Figure 5.4: A star with ten nodes

The resource consumption for all topologies look almost the same over all parameters as shown in figure 5.5.

Figure 5.5: The erratic behaviour is caused by the ”waiting” time between the rounds

The reason for this behaviour is, by forcing them to behave similarly. Because of the condition that every node needs at least half of the total number of nodes as neighbors, it does not really matter which topology is being chosen. In the end they are being forced to add new neighbors until they reach the threshold. With the similar amount of connections, the training process will also look similar.

Figure 5.6 shows that the choice of topology also has no effect on the performance of the model. This is because of the forced connection mentioned above. Because of the algorithm transforming the initials graphs into almost equal looking graphs afterwards, it was to be expected, that the they perform similarly.

Even though they behave similar, there are some outliers in the experiments as shown in figure 5.7.

In a fully connected topology every node has the maximum amount of connections. This means in a network with n nodes, every node has n-1 connections. The total number of nodes in a fully connected topology is always n*(n-1)/2. This is a huge increase of resource consumption. The benefits from a fully connected topology is a more stable setting overall. As long as there are more honest nodes than malicious nodes (which is the assumption), then the majority will always in the honest nodes' favor. As discussed

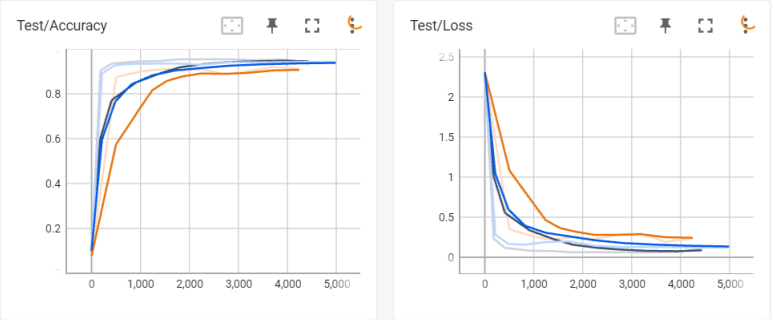

Figure 5.6: The choice of the topology does not have an significant impact

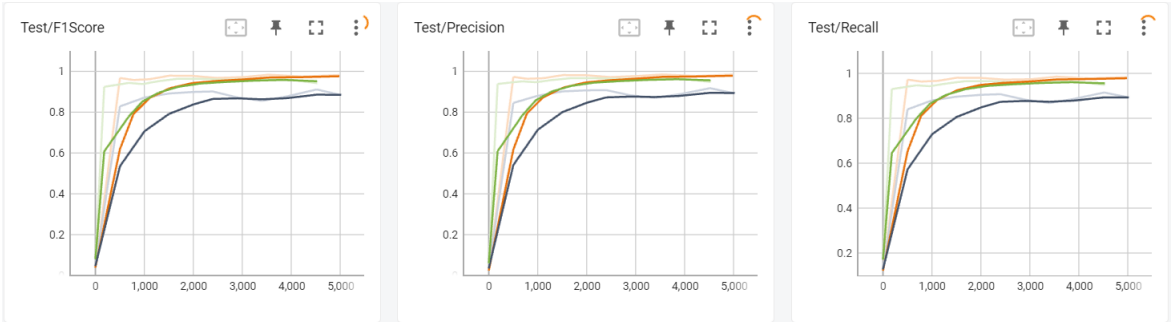

Figure 5.7: This example stands out the most. With the black one being the random topology, the orange being the star and green being ring, the difference is around 5 percent

before, the optimal number of connection is not a fully connected graph. This is because the number of honest nodes decreases with each addition of a honest node as a neighbor. So with each further addition of a honest node to the neighbors list, the chance of getting a malicious node increases as well, to a certain point where there are equal number of honest nodes and malicious nodes. At this point it is not better to add anymore nodes. So in theory, the fully connected graph performs worse. But in practise, it would require the knowledge of the number of malicious nodes as well as knowing which nodes have been added. This is no a realistic assumption, therefore a fully connected graph can still perform better as it is more stable, because the ratio of honest to malicious nodes is always the same, where as the ratio with the algorithm varies from run to run.

5.4 Case Study 2: Size

In this setting the size variable is being investigated. As per usual there are ten nodes, the poison amount is small (5 percent) and the topology is random. This time the number of nodes are the following:

- First run: 6 nodes
- Second run: 10 nodes

- Third run: 16 nodes

The poison amount is a percentage variable. The more nodes there are the more are also infected(in absolute numbers). But it is always going to be 5 percent of the total of nodes. Now it is safe to say that the resource consumption will increase drastically as with each node added, a lot more neighbors have to be added. It is also safe to assume that the reverse is true. Figure 5.8 shows the traffic between the scenarios. Because such implementation is not done on the fedstellar framework, as it needs the whole graph as an input already, this will only be done theoretically and calculate the cost from here. For a network of 6 nodes, the approximately number of new nodes added to the network after a simulation of 1000 graphs are 6.8. For a network of 10 nodes, the approximately number of new nodes added to the network after a simulation of 1000 graphs are 15.6. For a network of 16 nodes, the approximately number of new nodes added to the network after a simulation of 1000 graphs are 34.8. Depending on the cost of the fedstellar framework for adding a new connection, the cost will be added to the resource consumption.

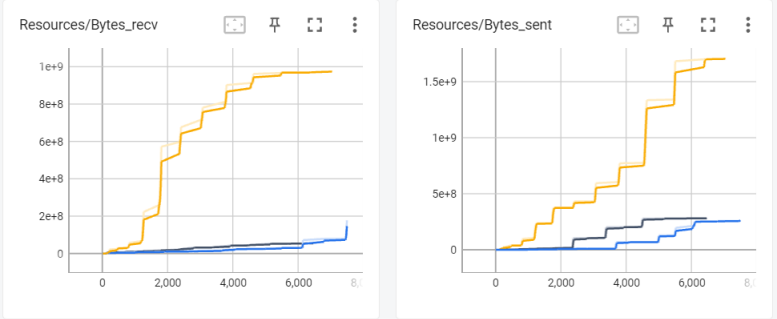

Figure 5.8: The resource consumption with 16 is huge. The jump is much bigger between the small and medium sized ones. As expected the resource consumption probably increases exponentially or near that trend

This is to be expected as with nearly double the amount of nodes, more than twice the connections are being added to the network. The jump from 10 to 16 is even larger as with 50 percent increase of nodes, the connection amount is being doubled.

Even with different sizes of the network the performance does not really suffer or benefit from it in any way. The differences in the performance are not really big. Note, that not all participant perform the same. Also the poison attack type is random, so it is expected to have some variance. So it is important to note that it only cost more resources to get bigger networks but it still has the same performance. In this case it would be preferable to have a small network with as little as possible number of participants. The conclusion is that the algorithm works on all sizes of networks, having a more significant impact on larger networks. Increasing the size of a network does not increase or decrease the performance significantly.

5.5 Case Study 3: Poisoning Type

There are 2 poisoning type. Data poisoning and model poisoning. Data poisoning leads to wrong results during the training phase as well as sharing the wrong results with ones neighbors. Model poisoning leads to change of model parameter, such that it does not behave as intended anymore. In both cases they reduce the performance of the overall system. In case of no algorithm applied to the system, the more poison introduced into the system, the worse its performance gets. Figure 5.9 shows a comparison with 6 nodes.

Figure 5.9: The poison type does not matter as they perform similarly

In all scenarios there were no difference whether the data or the model has been poisoned. They all performed equally. Following statistics in figure 5.10 and figure 5.11 are from scenario with 10 and 16 nodes respectively.

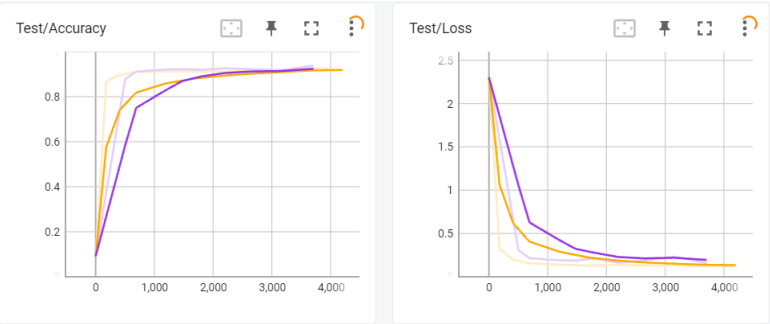

Figure 5.10: Performance of both scenarios are almost equal with ten nodes

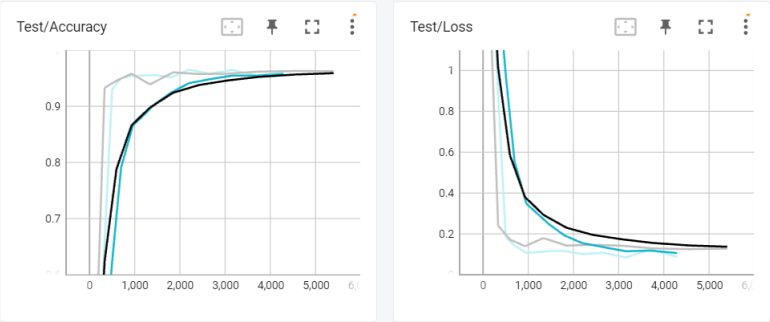

Figure 5.11: Even with larger networks no significant changes can be found

The poison type does not matter because the algorithm did not change the way the models share their parameters or implemented a check for the data. With only the change of the neighbors, whom they share their weighs with, the result of their training remains the same. Whether they are poisoned by data, or model, the result remains the same, which is that they are sharing poisoned parameters. Because the algorithm does not differentiate between the poison type. the results will be the same. The only factor that influences the outcome is the impact of the poison on the fedstellar model itself. There could be a chance, that the model is more susceptible to one poison type, but that is beyond the control of this algorithm.

5.6 Case Study 4: Poison severity

By varying the poison severity the impact of the algorithm can the tested, on different basis. Here the non poisoned setup is being compared with the poisoned setup on 3 different poison strength, to see if the algorithm performs worse or better based on the poison strength. In theory the algorithm enforces more connection, so it should lessen the impact of the poison on non poisonous nodes with poisonous neighbors. The algorithm has no impact on already poisoned nodes because it only manages the neighbors, so that the aggregation is not affected as much as it would be because it will have less poisoned update parameters on average. As discussed in case study 3, the poison type does not matter, so only consider data poisoning is being used here. Figure 5.12 shows the scenario with little poison.

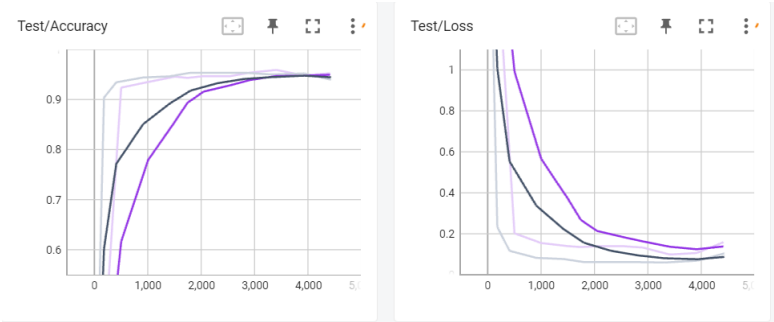

Figure 5.12: Black: With algorithm, purple: without algorithm

With little poison the overall accuracy does not suffer. But there is impact of the poison in the loss chart. As there is some poison, the higher loss on the poisoned side occurs, due to the label flipping inferring with the training such that now and there the machine learning model completely misses the point. Other than that the impact of the poison is not strong enough even in the base scenario.

The second scenario is with 10 percent poison. The expectation is at least to see some difference between the two scenarios otherwise the algorithm did not have an impact.

There is a substantial amount of difference in the accuracy as well as in the loss perspective in figure 5.13.

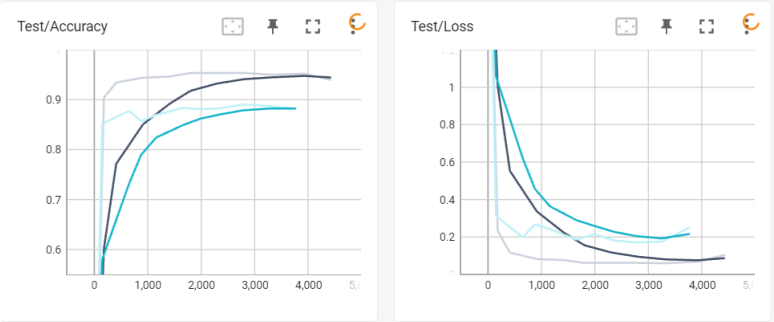

Figure 5.13: Black: with algorithm, purple: without algorithm

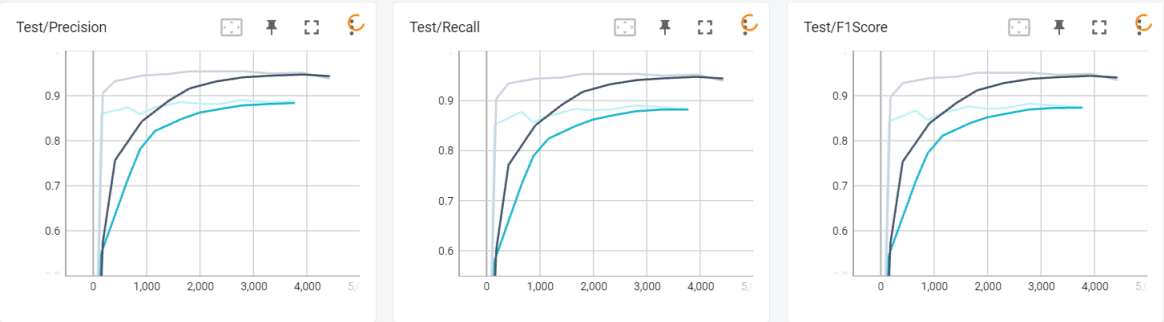

Figure 5.14: Black: with algorithm, Light blue: without algorithm

Even the precision and recall values indicate, that the algorithm has an impact on the overall performance of the system with the algorithm applied to in figure 5.14.

The last scenario is with 20 percent poison.

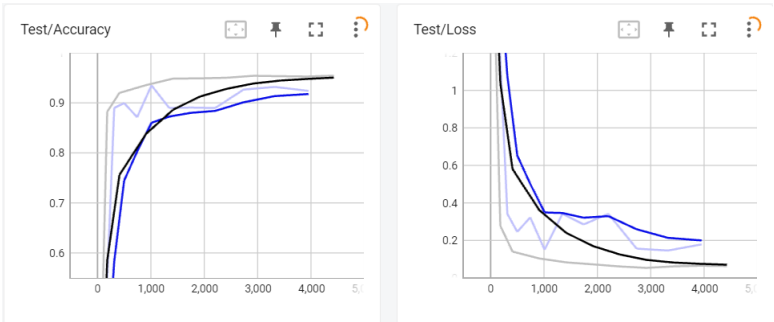

Figure 5.15: Black: with algorithm, Blue: without algorithm

The results in figure 5.15 do meet the expectations. In this category of poison amount the algorithm does have a visible impact on accuracy and loss. What really stands out is that the machine learning model with medium poison has a worse accuracy than the model with large amount of poison in it. This is only the case if choosing those exact two models for comparison. For example in this scenario in figure 5.16 the order of performance is back to ”normal”.

The conclusion for this experiment is that while it had no impact on lower severity of poison it lessened the impact in presence of larger amount of poison. Because of DFL

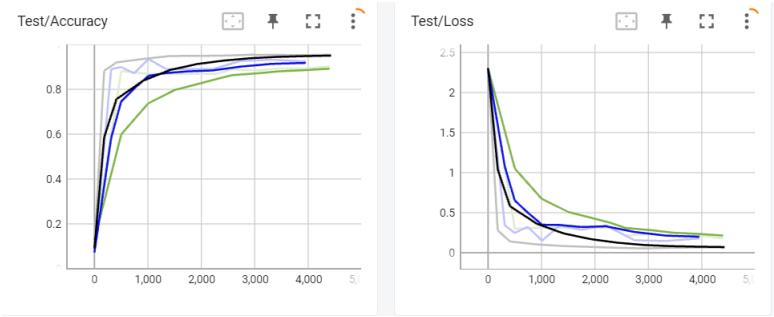

Figure 5.16: The average performs as expected

though, it is still depending on the view of the participants. Some of the participant perform worse because the poison struck there first. The center of the cause will perform worse while getting further away from it will increase the performance. The algorithm decreases the radius in which the performance drops significantly.

5.7 Case Study 5: Topology and poison severity

This case study considers two parameters at once, by combining topology and the poison severity. With the intention to examine the impact of various poison strength on topology such as star and ring. The expectation is that there is little to no difference between the topologies as the algorithm forces more connection between nodes. There is actually one case where adding in more neighbors could be worse. That is, when there is a ring formation. By adding more neighbors, there will be essentially a shortcut for the poison to spread through the system. But the topology will also gain more neighbors for parameter sharing, so in the end it should not worsen the performance in presence of little poison amount. In higher severity of poison this could worsen the performance as there is too much poison spreading too much negating and even surpass the effect of adding neighbors.

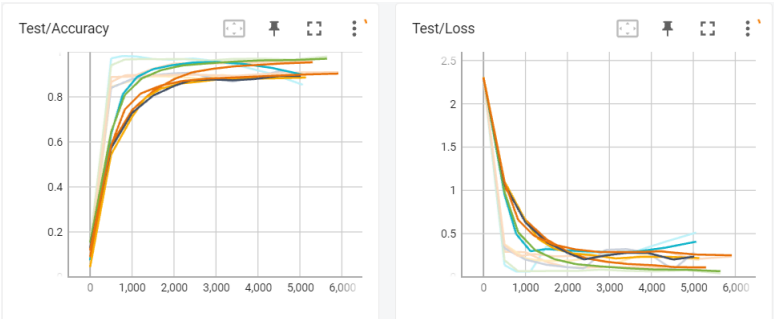

Figure 5.17: All combination of poison severity and topologies in on display.

Neither the poison amount and the topology really matter as shown in figure 5.17. They all are more or less in the same direction and have some variance, because of the single participants view, but in general there is not much unexpected. Because of the randomness of adding neighbors, if there are no nodes anymore matching the criteria and the special

topology, some nodes do have much more connections than others. This leads to some differences sometimes and figure 5.18 shows an example of it.

Figure 5.18: The effect of selecting single participants of a network

The difference is about 5 percent. But what really is strange is that the pink graph is representing the performance of a model with large amount of poison in it. To test if this is only an outlier or not, the same settings as above have been tested several times. The result was that it was an outlier. In the end the amount of poison did not even matter. They all scatter around the high 80 to 90 percents as shown in figure 5.19.

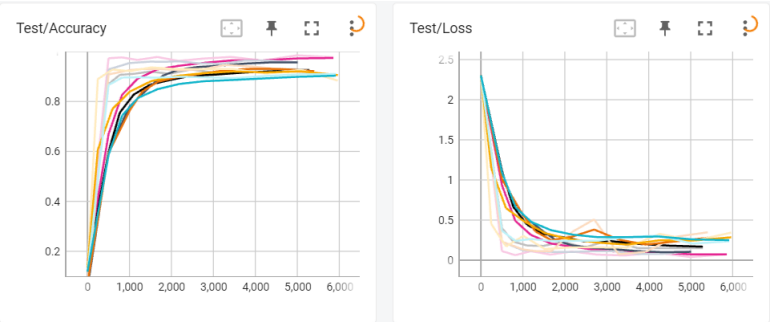

Figure 5.19: With enough number of experiment the result are in order again with some outliers

This is an interesting observation as from the previous setting with poison severity, the poison amount mattered in the medium and large poison amount scenarios, whereas in here it does not matter. Because of the special topologies, for example the star, every node only has one neighbor at the beginning. So with the algorithm, they add a lot more neighbors than the usual random topology because in a random setup the nodes do have more than one neighbor on average. And due to the restriction of adding neighbors of a certain distance, especially in a star topology, every node is a distance of two away from each other. So after the algorithm the network becomes almost fully connected and the distance between the nodes is at max two. This makes them really close to each other meaning the poison effect gets mitigated so heavily that it cannot have an impact unless the poison amount is close or more than half of the nodes. The ring topology has the property, that every node has two neighbors. So by forcing a connection of a certain distance, the node will always find two additional neighbors of the distance, because it is a ring. After that it will add one random node. From this the distance between the nodes cannot be further than two in a network of 10 nodes. This also means that the node are

relatively close to each other dampening the poison effect. Compare this to a random topology where for example, it has a lot of long strings. Here it is much more likely to get nodes of distance more than two. Note that for this section more runs are needed for a conclusion. These result could be from a chain of topologies which have been formed by the algorithm, that coincidentally lead to this.

Chapter 6

Summary and Conclusions

6.1 Conclusion

The main research point in this thesis was to find a new way to enhance byzantine robust aggregation rules. By changing the way of sharing the parameters of the participants in A decentralized federated learning setting, this thesis increased the resource consumption overall but also increased the accuracy, precision and recall in some scenarios. Because decentralized federated learning had not gotten that many attention yet, this thesis therefore addressed the lack of materials in this regard.

The theoretical background served as an introduction to machine learning as well as an introduction to federated learning. It was necessary to understand how they work, and more importantly how they differ from each other. With the background knowledge the aggregation rules had been analyzed to find their usage, advantages and disadvantages. Based on the analysis a new way of enhancing byzantine robustness had been proposed. The main idea behind the the algorithm was to increase the number of weights or model parameters from other participant to balance out the poisoned one. This was especially important for participants with only one neighbor as they could not tell if the neighbor was poisoned or not. This for example, made fedavg really difficult to use, as averaging over 2 model parameter where one of the could be poisoned, was not a good idea. So this algorithm added more neighbors for such cases. Settings like the amount of neighbors to be added or which neighbors to consider, could be modified to a certain extend. A threshold allowed it to set the number of neighbors required for each participant and the distance parameter allowed it to control the selection to certain degree.

The implementation was done on a given framework. The framework was a fully operational decentralized federated learning model, in which settings like topology, aggregation rule, poison degree and so on could be set run an experiment. Because the framework required everything to be set at the beginning, the modification of its ”starting file” had to be modified by adding new rules to it before it started the whole training process. In this case the incidence matrix represented the network input at the beginning. So by changing this matrix according to the algorithm, a new matrix will serve as the matrix to be used for training. With this modified version of the framework multiple experiment have been

done based on a variety of criteria. Poison severity, number of participants, topology and much more parameters were important to separate the experiment in order to get good and wide covered results. In conclusion, the evaluation proved that the algorithm worked in most cases but not all. Additionally the selection of the neighbors were not optimal, for example, some participants got more than necessary numbers of neighbors.

6.2 Limitations and Future Work

As of now, the implementation of the algorithm takes place before the training process on the fedstellar framework. It is static and once the modification has been done, it cannot be changed anymore during the training period. So if a change is needed, because the topology changes during the training, then this would not be possible. As mentioned before, the algorithm does have weaknesses in several areas. The experiment for example, only ran up to 16 participants. A more thoroughly series of experiment need to be done on more situations. This could be a hard deciding factor for the algorithm as each node we add, increases the resource consumption more than linear. In general more experiment have to be done on this setting as we were limited by the capacity of our local machines. The resource consumption is a important point in this work. An algorithm that works properly, but can not be run in foreseeable future, because its complexity is too high, is not really useful. This is why it is important to reduce as much redundant computation as possible should be aimed for. At the current state, the algorithm does not balance the distribution of neighbors. It means that if for a node it has the choice between 2 nodes to add as a neighbor, it will pick randomly. This leads to some nodes having (much) more neighbors than the threshold which leads to longer computation time during the training. The question of if this is a bad or good thing has to be investigated further. Also if the algorithm cannot find anymore nodes with the indicated distance to add, it will start add randomly. This is also a point where improvement can done on, for example looking for the nodes with the least number of neighbors at the moment. By doing so there arise other problems such as what if the distance of that node is not high enough. Considering all of this the algorithm satisfies many conditions, but can also use some improvement on it.

Bibliography

[1] Y. Liu, T. Han, S. Ma, *et al.*, *Summary of chatgpt/gpt-4 research and perspective towards the future of large language models*, 2023. arXiv: 2304.01852 [cs.CL].

[2] IBM. "What is machine learning?" (), [Online]. Available: https://www.ibm.com/topics/machine-learning. (accessed: 04.06.2023).

[3] diffen. "Data vs. information". (), [Online]. Available: https://www.diffen.com/difference/Data_vs_Information. (accessed: 04.06.2023).

[4] IBM. "What is data labeling?" (), [Online]. Available: https://www.ibm.com/topics/data-labeling. (accessed: 04.06.2023).

[5] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning", in *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, M. Cord and P. Cunningham, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–49, ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_2. [Online]. Available: https://doi.org/10.1007/978-3-540-75171-7_2.

[6] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Machine learning: A review of classification and combining techniques", *Artificial Intelligence Review*, vol. 26, pp. 159–190, Nov. 2006. DOI: 10.1007/s10462-007-9052-3.

[7] A. O. Sykes, *An Introduction to Regression Analysis*. 1993.

[8] M. Andrychowicz, M. Denil, S. Gómez, *et al.*, "Learning to learn by gradient descent by gradient descent", in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf.

[9] H. Barlow, "Unsupervised Learning", *Neural Computation*, vol. 1, no. 3, pp. 295–311, Sep. 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.3.295. eprint: https://direct.mit.edu/neco/article-pdf/1/3/295/811863/neco.1989.1.3.295.pdf. [Online]. Available: https://doi.org/10.1162/neco.1989.1.3.295.

[10] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Selection of k in k-means clustering", *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103–119, 2005. DOI: 10.1243/095440605X8298. eprint: https://doi.org/10.1243/095440605X8298. [Online]. Available: https://doi.org/10.1243/095440605X8298.

[11] Y. Li, *Deep reinforcement learning: An overview*, 2018. arXiv: 1701.07274 [cs.LG].

[12] C. O. S. Sorzano, J. Vargas, and A. P. Montano, *A survey of dimensionality reduction techniques*, 2014. arXiv: 1403.2877 [stat.ML].

[13] S.-C. Wang, "Artificial neural network", in *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US, 2003, pp. 81–100, ISBN: 978-1-4615-0377-4. DOI: 10.1007/978-1-4615-0377-4_5. [Online]. Available: https://doi.org/10.1007/978-1-4615-0377-4_5.

[14] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions", *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020. DOI: 10.1109/MSP.2020.2975749.

[15] Y. Zhou, Q. Ye, and J. Lv, "Communication-efficient federated learning with compensated overlap-fedavg", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 192–205, 2022. DOI: 10.1109/TPDS.2021.3090331.

[16] C. Hu, J. Jiang, and Z. Wang, *Decentralized federated learning: A segmented gossip approach*, 2019. arXiv: 1908.07782 [cs.LG].

[17] H. Kirrmann and K.-E. Großpietsch, vol. 50, no. 8, p. 359, 2002. DOI: doi:10.1524/auto.2002.50.8.359. [Online]. Available: https://doi.org/10.1524/auto.2002.50.8.359.

[18] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", *ACM Trans. Program. Lang. Syst.*, vol. 4, Feb. 2002. DOI: 10.1145/357172.357176.

[19] Z. TIAN, L. CUI, J. LIANG, and S. YU, "A comprehensive survey on poisoning attacks and countermeasures in machine learning", Ph.D. dissertation, University of Technology Sydney, Shandong Computer Science Center (National Supercomputer Center in Jinan), 2022.

[20] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems", Ph.D. dissertation, Georgia Institute of Technology, 2020.

[21] Y. Fraboni, R. Vidal, and M. Lorenzi, "Free-rider attacks on model aggregation in federated learning", Ph.D. dissertation, Université Côte d'Azur, Accenture Labs, 2020.

[22] X. Zhou, M. X. 1, Y. Wu, and N. Zheng, "Deep model poisoning attack on federated learning", Ph.D. dissertation, Hangzhou Dianzi University, 2021.

[23] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent", Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2017.

[24] Y. C. Dong Yin, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates", Ph.D. dissertation, Cornell University, 2021.

[25] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping", Ph.D. dissertation, Duke University, The Ohio State University, 2021.

[26] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium", Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2018.

[27] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, *Scaffold: Stochastic controlled averaging for federated learning*, 2021. arXiv: 1910.06378 [cs.LG].

[28] N. Shi, F. Lai, R. A. Kontar, and M. Chowdhury, "Fed-ensemble: Improving generalization through model ensembling in federated learning", Ph.D. dissertation, University of Michigan, 2021.

[29] G. Hinton and J. D. Oriol Vinyals, "Distilling the knowledge in a neural network", Ph.D. dissertation, University of Toronto, Canadian Institute for Advanced Research, 2015.

[30] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning", in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf.

[31] D. E. Knuth, *The TEX Book*. Addison-Wesley Professional, 1986.

[32] M. Lesk and B. Kernighan, "Computer typesetting of technical journals on UNIX", in *Proceedings of American Federation of Information Processing Societies: 1977 National Computer Conference*, Dallas, Texas, 1977, pp. 879–888.

[33] E. T. M. Beltrán, Á. L. P. Gómez, C. Feng, *et al.*, *Fedstellar: A platform for decentralized federated learning*, 2023. arXiv: 2306.09750 [cs.LG].

List of Figures