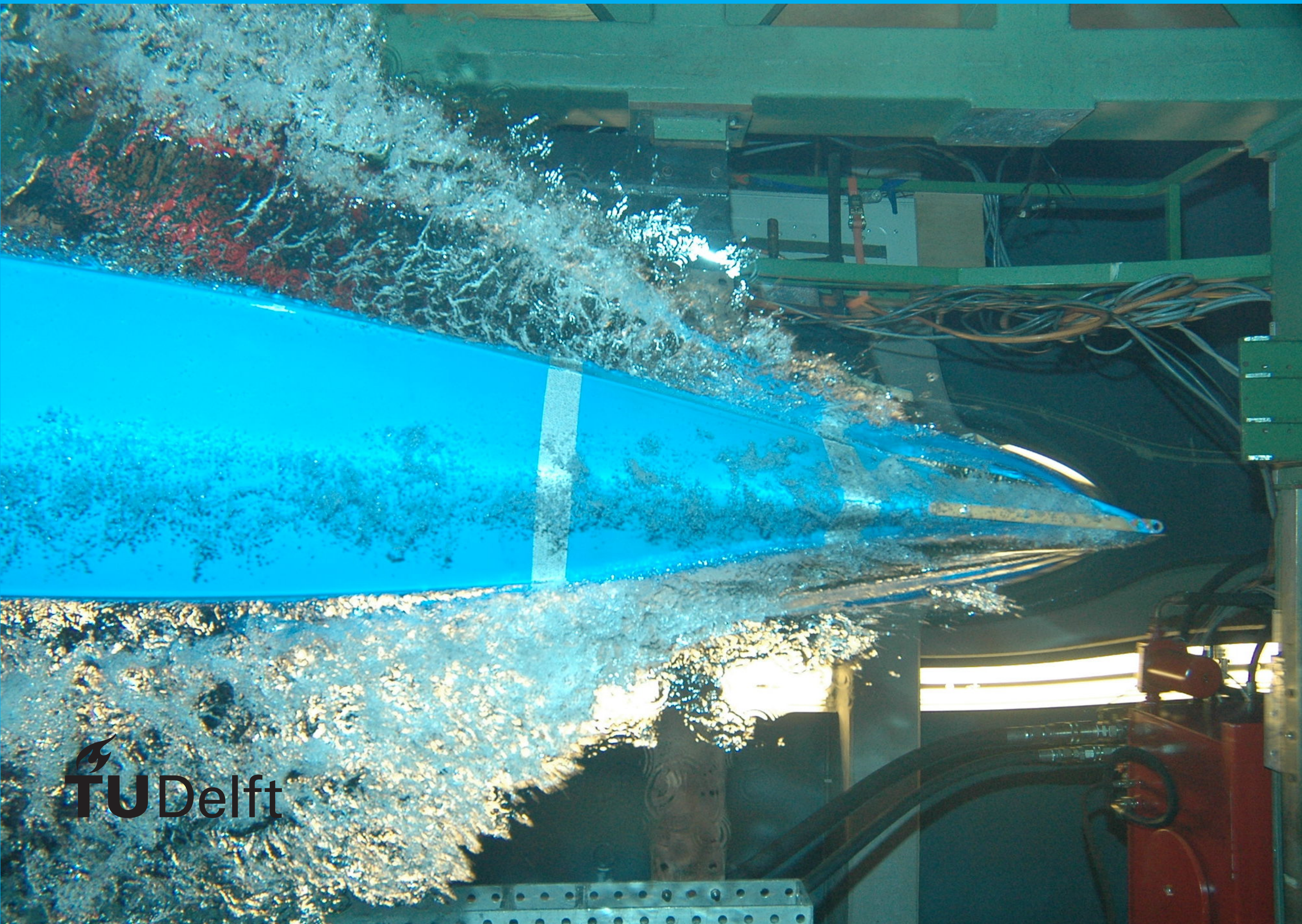
A comprehensive comparison between federated and centralized learning

S.J.F. Garst



A comprehensive comparison between federated and centralized learning

by

S.J.F. Garst

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday December 22 2021 at 10:00 AM.

Student number:	4484320	
Project duration:	May 24, 2021 – December 22, 2021	
Thesis committee:	Prof. dr. ir. G. J. T. Leus,	TU Delft, thesis supervisor
	Prof dr. ir. M. J. T. Reinders,	TU Delft, daily supervisor
	Dr. ir. D. M. J. Tax,	TU Delft

TU Delft

Preface

This document describes my graduation project in order to obtain the master degree in Electrical Engineering at the Delft University of Technology. When offered a topic with which my supervisor had limited experience, I realised that moving into unknown territories is something that makes me tick. This was the last in a series of events over my student career in which I got to know myself, a vital activity which I think the culture at the TU Delft encourages.

The graduation process itself went about as smoothly as I could have ever hoped, starting with a planning on day one which we were able to follow until the end. For that, my deepest graditude goes out to my supervisor, Marcel Reinders, whose guidance was the enabling factor to such a smooth process. I also want to thank everyone that was involved in the technical discussions, in particular Julian Dekker, David Tax and Geert Leus. You were able to bring up new ideas, growing the project even further. Finally, I want to thank everyone who I was able to bounce ideas off of during my graduation. This includes my parents, friends and roommates; It is incredible what difference it makes to be rambling at another person instead of your own computer screen.

S.J.F. Garst
Delft, December 2021

Contents

1 Introduction

Nowadays lots of data is available for the use of machine learning applications. However, in some use cases this data does not naturally reside at a single location, and centralizing said data might be difficult or straight up impossible due to regulation. One example is that of medical data [29] [5], which resides at different hospitals or medical institutions, but which cannot leave said institutions due to privacy concerns.

In order to still try to leverage this type of data, the concept of federated learning [16] was introduced. Instead of gathering the data at the place where a model is being trained, in a federated learning environment, the model gets send to wherever the data is available: the so-called clients. At these clients, the model undergoes some form of training, after which the updated model is sent back to a central point, referred to as the server. The server then aggregates all the local updates in order to create a new global model, which it then sends back to all of the clients, and the cycle repeats. With this setup, only model updates are being communicated, and since the original data never has to leave its origins, the entire process becomes less privacy-sensitive.

Formally, the federated learning problem definition can be described as follows: given n clients, minimize the function

$$\underset{W}{min}\, F(W(X,y)) = \sum_{i=1}^{n} L(W(x_i), y_i) \tag{1.1}$$

where L is a loss function,W a set of model parameters (for example the weights and biases of a neural network) and X = $\{x_i\}_1^n$, with x_i the dataset on client i with its corresponding labels y_i.

The concept of federated learning is adjacent to other fields trying to learn from distributed data, particularly distributed learning [30] [18]. Distributed learning broadly operates similar to federated learning, having a network of clients trying to solve an optimization problem. However, distributed learning allows for different learning objectives per client, which are only locally known. This is contrary to federated learning, where everyone utilizes the same loss function, but on data which is only locally known. Besides, federated learning uses a star network, with a centralized server connected to each client. Such a central point is not present in distributed learning, with clients having only connections to (some) other clients. A more in-depth analysis of distributed learning is given in section 2.1. A variant of federated learning which utilizes a single loss function, but without a central server and thus a network architecture much more akin to distributed learning, has been coined swarm learning [26].

Since the conception of federated learning back in 2015 [16], lots of research has been done on its efficiency, performance and privacy-preserving properties [11] [12]. Communication ends up being a bottleneck in many federated systems [13]. As a result, development of techniques to reduce communication rounds have been emerging [9] [4].With regards to performance, many analyses have been made on the performance of the original federated algorithm, called federated averaging (details in section 2.2.1). These researches focus usually on preformance under poorer data distributions [7] [3] [6], as it can be shown that, under certain assumptions about the data distribution, federated averaging gives results equivalent to stochastic gradient descent, see appendix A.2. As a result, extensions of federated averaging trying to accommodate for different (non-IID) data distributions have been developed, e.g. [6] [23]. Privacy preservation has been explored by means of constructing specific attacks on federated systems [2] [25]. As a response, extensions on the original federated algorithms which include some form of increased privacy preservation is becoming a vast area of research [27].

Although all of the aforementioned researches have developed the federated learning concept into research areas of their own, many of the analyses remain mostly theoretical. Besides, it is usually taken for granted that the federated approach is a given, while there are certain use cases where the question whether or not to use federated learning is an important design decision. In these cases, for example medical research, where it is possible, albeit tedious, to gather enough data in a central framework, it remains a tradeoff for which the performance differences between a central and federated model are an important factor. However, to our knowledge, experimental results comparing federated to centralized models has been scarce. Therefore, in this paper, we will describe a vast set of experiments in which centralized and federated models are compared with one another. We explore the use of different classifiers on multiple datasets, distributed in multiple ways. In doing so, we hope to shed some light on different scenarios in which federated learning might perform similarly to a centralized model, and when to be careful in assuming such a similar performance.

2
Background

2.1. Distributed Learning

The goal of distributed learning is to minimize the sum of a set of local cost functions in a network of n clients. This can formally be written as

$$\underset{x}{min}\, F(x) = \sum_{i=1}^{n} f_i(x^i) \tag{2.1}$$

Here f_i is the local cost function of client i, which is only known by itself. Note that these functions can differ between clients, a vital difference compared to federated learning. Furthermore, x is a resource vector, with its components the resources allocated to each client.

The distributed learning setting does not utilize a central server, and the network is not a complete graph either. Therefore, a notion of neighbours exist. Each round, each client can exchange information with each of its neighbours only, and will try to decrease its cost function based on said information. For this purpose, a weight matrix A is being updated by each client. This matrix holds nonzero weights for each of its neighbours, including itself, and zero weights for all other clients in the network. The total weights add up to 1, and their magnitude is a measure for the value of the information of the corresponding neighbour. A seminal algorithm, being distributed Stochastic Gradient Descent (dSGD) [18], uses this matrix as follows:

$$x_{r+1}^{i} = \sum_{j=1}^{n} A_r^j x_r^j - \eta_r^i * \nabla f^i(x_r^i) \tag{2.2}$$

Where η_r^i is the learning rate at client i on round r, and $\lambda f^i(x^i(k))$ the local gradient over the cost function at client i on round r. Note that, in case of no neighbours, the A-matrix becomes all zeros except for the value corresponding to a clients own information, wich will become one; This retrieves the standard SGD used in non-networked machine learning applications.

From the basic model described by equation 2.2, several extensions have been made over the years [30]. In the original work, distributed SGD utilizes a learning rate which is variable, and diminishes over time. Another approach is to keep the learning rate fixed, but to utilize more information than just the gradient on the cost function from the latest time step. That is, to look at more previous iterations and their respective cost functions.

One of these algorithms is EXTRA (EXact firsT-ordeR Algorithm) [24], which uses the two previous iterations of the cost function instead of just one. Formally, this becomes:

$$x_{r+1}^{i} = x_r^i + \sum_{j=1}^{n} A_j^i x_r^j - \sum_{j=1}^{n} \tilde{A}_j^i x_{r-1}^j - \alpha(\nabla f^i(x_r^i) - \nabla f^i(x_{r-1}^i)) \tag{2.3}$$

Where $\tilde{A} = \frac{I+A}{2}$ with I the identity matrix. There is the exception of r=0 as there only $x^i(0)$ can be used, thus the terms using x_{r-1}^i are left out for the first iteration.

Besides use of more previous iterations, another option is to modify the gradient calculation method. The DIGing (Distributed Inexact Gradient method and the gradient trackING technique) [21] [19], a separate state which corresponds to the gradient estimation is being updated. This state is also being shared with neighbours, in a similar way to the x^i state.

$$x^i_{r+1} = \sum_{j=1}^{N} A^i_j x_j(r) - \alpha y^i(r) \tag{2.4}$$

With y^i gets updated as:

$$y^i_{r+1} = \sum_{j=1}^{N} A^i_j y_j(r) + \nabla f^i(x^i(r+1)) - \nabla f^i(x^i(r)) \tag{2.5}$$

2.2. Federated Algorithms

2.2.1. Federated Averaging (fedAVG)

The main goal of federated averaging was to create the option to learn over multiple datasets without the need to upload said data to a central point. This is achieved by learning a model on each of these smaller datasets locally, and then sharing the model parameters with the other models.

The locations of the data, which is where the local models will be learned, are called ’clients’. These clients are connected to a central point, named the ’server’. The server maintains a global model, which it updates according to the results of the updates at the clients.

The algorithm is divided in rounds. Each round starts with the server sending out the global model to all clients. Each client then makes one or more local update steps in which it updates the model parameters to lower its local loss function $f(W(X^i), y^i)$. Once completed, it sends these locally updated parameters back to the server. Once the server has received an update from all clients, it aggregates these local parameters to create a new global parameter set, which it sends out in the next round.

The local update step is driven by Stochastic Gradient Descent (SGD). The authors of fedAVG [16] introduce both batch learning and multiple learning epochs per communication round, so the local update becomes:

$$W^i_{r+1} = W^g_r - \sum_{e=1}^{E} \sum_{b=1}^{B} \eta \nabla L(W^g_{r,b,e}(X^i_b), y^i_b) \tag{2.6}$$

Where E is the amount of local epochs and B the batch amount. During all experiments in this paper, E has been kept at 1. B was kept at 1 except for all experiments using the CNN (curiously, without batch learning, the federated CNN would not converge).

After the local update has been completed on all participating clients, the server starts the global update section of the current round. This global update simply consists of a weighted average of all the received. The original creators of federated averaging [16] gave the option of sampling clients, i.e. not using all clients for every update step. The motivation behind this was that the performance increase becomes negligible after a certain amount of clients:

$$W^g_{r+1} = \frac{1}{S} \sum_{i=1}^{C} s^i * W^i_{r+1} \tag{2.7}$$

Where W^g_{r+1} is the new set of global parameters, and X_i is the set of parameters which have been sent back by client i earlier this round. s_i is the size of the dataset on client i, and $S = \sum_{i=1}^{C} s_i$. C is the total amount of participating (sampled) clients in a round. According to [16], sampling becomes useful when using more than 100 clients. As all experiments in this paper use ten or less clients, no sampling was used in the experiments described within this paper. To summarize, there are three parameters that tune the behavior of federated averaging:

Parameter	Meaning
B	batch size
C	fraction of sampled clients
E	local epochs

Two of these, E and C, have been kept at 1, and B has only been altered during the experiments with the Convolutional Neural Network.

2.2.2. SCAFFOLD

The SCAFFOLD algorithm was created as a means to increase the performance of the earlier discussed federated averaging, specifically with respect to a non-IID setting. A non-IID setting means that the distribution of classes between different clients is not similar, i.e. some clients only having access to a very low amount (in some cases even zero) of samples of certain classes. The problem that arises in federated averaging within the non-IID setting is that a drift is starting to settle in, meaning that the global parameters do not or slowly converge to the optimum. In order to combat this drift, the authors of [6] introduce a new algorithm for Stochastic Controlled Averaging, called SCAFFOLD.

The main addition in SCAFFOLD is the introduction of a control variate for all clients, *and* for the server. This control variate consists of a set of values with the same structure as the set of parameters (i.e. it has one value per parameter). Intuitively, it denotes the direction (and magnitude) of the local update of said parameter. If this direction is different from many other clients, the control variate is used to compensate for that difference, which decreases the aforementioned drift.

The general setup of SCAFFOLD is similar to that of federated averaging. The algorithm consists of rounds, which consists of a global part at the server, and a local part which happens at all clients simultaneously. at the beginning of each round, the server sends the global model to all clients, as well as its own control variate c. Now, each client makes a pass over its local data to calculate the gradient over its loss function. This also happens in federated averaging, as part of the stochastic gradient descent step. After the gradient has been determined, the parameters get updated as follows:

$$W^i_{r+1} = W^g_r - \eta \nabla L(X^i_{train}, y^i_{train}) + c^g_r - c^i_r \tag{2.8}$$

Note that if both c^g and c^i are all zero, equation 2.8 becomes standard SGD, and SCAFFOLD becomes federated averaging (at least for the local step). After the model has been updated, each client also needs to update its control variate. This is done according to the following equation:

$$c^i_{r+1} = c^i_r - c^g_r + \frac{1}{\eta_l} W^i_r - W^i_{r+1} \tag{2.9}$$

Finally, W^i_{r+1} is sent back to the server, which denote the end of the work on the client for the round. Once the server has received W^i_{r+1} for all $i \in N$, it aggregates the local updates into a new iteration of the global model. This update is executed as follows:

$$W^g_{r+1} = W^g_r + \frac{\eta_g}{N} * \sum_{i=1}^{N} W^i_{r+1} - W^i_r \tag{2.10}$$

Which is equivalent to the global update of federated averaging, if η_g is set to 1 (which is the case during all experiments described in this report). Different from federated averaging is the need to update c as well, which is done according to the following equation:

$$c^g_{r+1} = c^g_r + \frac{1}{N} * \sum_{i=1}^{N} (c^i_{r+1} - c^i_r) \tag{2.11}$$

Once this is done, the next round starts by the server sending out the updated model parameters and c. A big difference wich federated averaging is that SCAFFOLD is a *stateful* algorithm, with the control variates functioning as some sort of state for clients and server.

2.2.3. inPrivate Learning

Whereas federated averaging and SCAFFOLD are quite similar in nature, the inPrivate learning algorithm is quite different. The global idea of using rounds in which something happens on the clients and on the server sequentially still exists, but also covers the extent of the similarities between the algorithms.

The idea behind inPrivate learning is to implement Gradient Boosting Decision Trees (GBDT) in a federated manner. To accomplish this, clients construct trees sequentially, with just one client constructing only one tree each round.

The main gain of using the federation, compared to just creating a local model on the (small amount of) available data, is the use of ’gradient boosting’. Gradient boosting can be used to combine an ensemble of ’weak learners’, such as decision trees. Intuitively, gradient boosting means that, after the first learner, all subsequent learners try to correct for their predecessors, thereby iteratively lowering the total loss over the training dataset.

In InPrivate learning, this is accomplished as follows: Once a client receives the previously made decision trees (the first round is an exception, there is no boosting yet, as there is nothing to boost off), it calculates the gradient of its loss function according to these previously constructed trees. It then creates its own tree such as to lower this gradient.

3 Methods

3.1. Algorithm overview

In total, five different classifiers were implemented in a federated setting: A Logistic Regressor (LR), a Support Vector Machine (SVM), a Fully connected Neural Network (FNN), a Convolutional Neural Network (CNN) and a Gradient-Boosting Decision Tree protocol (GBDT). Except for the GBDT, all classifiers follow the same algorithm, which is given by pseudocode 1. The algorithm for the GBDT is given by pseudocode 2. Notation used in these pseudocodes can be found in table 3.1.

First, a brief description of algorithm used by the first four models is given. First, a model is initialized on the server with random values. this model is sent to all clients. Then every client (in parallel) performs a local training step to update its coefficients by means of Stochastic Gradient Descent (SGD). All clients then send back their updated model to the central server. There, these models are gathered, and combined to update the global model. Finally, the next epoch starts, and the combined coefficients from the previous round are now used as the new values sent out.

Table 3.1: notation overview

Notation	Meaning
W_r^g	global model at round r
W_r^i	model at client i at round r
X_{train}^i	training data at client i
Y_{train}^i	labels for trainig data at client i
X_{test}^i	test data at client i
Y_{test}^i	labels for test data at client i
acc_r^i	accuracy score of client i at round r
s_i	dataset size (amt of samples) at client i

Algorithm 1 the federated implementation of all classifiers except for GBDT

```
1:  init $W^g_0$ as random
2:  for all r rounds do
3:      On server: Send $W^g_r$ to all clients i
4:      On clients: $W^i_r \leftarrow W^g_r$
5:      On clients: $acc^i_r = acc(W^i_r, X^i_{test}, Y^i_{test})$
6:      On clients: $W^i_{r+1} \leftarrow localStep(W^i_r, X^i_{train}, Y^i_{train})$
7:      On clients: $s^i \leftarrow size(X^i_{train})$
8:      On clients: send $W^i_{r+1}$, $acc^i_r$, $s_i$ back to server
9:      On server: retrieve $W^i_{r+1}$, $acc^i_r$ and $s_i$ for all $i$
10:     $W^l = \{W^1_{r+1}, W^2_{r+1}, .., W^i_{r+1}\}$
11:     $S = \{s^1, s^2, .., s^i\}$
12:     On server: $W^g_{r+1} = globalStep(W^l, S)$
13:     On server: $acc^i_r = acc(W^g_{r+1}, X_{test}, Y_{test})$
14: end for
```

The functions $localStep$ and $globalStep$ in lines 6 and 13 respectively are either an implementation of the federated averaging [16] algorithm or of the SCAFFOLD algorithm [6].
For the implementation of federated averaging, *localStep* is simply done by means of Stochastic Gradient Descent, i.e. :

$$W^i_{r+1} = W^i_r - \eta_l * \nabla L(X^i_{train}, y^i_{train}) \tag{3.1}$$

where $L(X^i_{train}, y^i_{train})$ is a loss function which differs between classifiers.

The global step of federated averaging consists of, as the name suggests, taking the weighted average for all the coefficients:

$$W^g_{r+1} = \frac{1}{N * S} * \sum_{i=0}^{N} s^i * W^i_{r+1} \tag{3.2}$$

Where s^i is the dataset size of the i$^{\text{th}}$ client, and $S = \sum_{i=1}^{N} s^i$.

For both the local and global step, SCAFFOLD expands on federated averaging by means of a control variate. for the local step:

$$W^i_{r+1} = W^i_r - \eta_l * \nabla loss_i + c^g_r - c^i_{r+1} \tag{3.3}$$

Both c^g and all c^i are initialized as all zero. For the second part of the local step, c^i gets updated as:

$$c^i_{r+1} = c^i_r - c^g_r + \frac{1}{\eta_l} W^i_r - W^i_{r+1} \tag{3.4}$$

A global update step of SCAFFOLD first updates the coefficients as follows:

$$W^g_{r+1} = W^g_r + \frac{\eta_g}{N} * \sum_{i=0}^{N} W^i_{r+1} - W^i_r \tag{3.5}$$

And then updates c :

$$c^g_{r+1} = c^g_r + \frac{1}{N} * \sum_{i=0}^{N} (c^i_{r+1} - c^i_r) \tag{3.6}$$

A more in-depth analysis of federated averaging and SCAFFOLD is given in chapter 2.2.1 and 2.2.2 respectively.

The federated GBDT is implemented differently from the other classifiers, as due to the nature of the GBDT model, it does not lend itself very well for parameter averaging such as in federated averaging. Instead, for the GBDT, we followed the "inPrivate Learning" algorithm [31]. In this algorithm, only one client is active per round; This is the client referred to as C_{active} in the pseudocode. This client uses the decision trees of its predecessors to calculate a loss on its own dataset, which it then uses to boost the decision tree it builds. A more in-depth analysis of this algorithm is given in section 2.2.3.

Algorithm 2 The implementation for the inPrivate learning algorithm

```
init $W^g$ as random
for all r rounds do
    $C_{active}$ = r mod N
    send $W^g$ to client $C_{active}$
    Do on client $C_{active}$:
        calculate loss on local training set
        create new tree using calculated loss to boost
        add tree to $W^g$
        $acc_r = accuracy(W^g, X_{test}^{C_{active}}, Y_{test}^{C_{active}}$
        send $W^g$, $acc_r$ back to server
    End
    Receive Y, $acc_r$ from client $C_{active}$
    $acc_i^g = accuracy(W^g, X_{test} Y_{test})$
end for
```

3.2. Datasets

All classifiers were tested on several datasets. These datasets were based on three different 'baseline' datasets: MNIST [10], fashion MNIST [28] and a dataset used for the prediction of Acute Myeloid Leukemia (AML) [26]. The MNIST dataset was transformed into both a two-class and a four-class problem. The AML dataset consists of three subsets: A1-A3. The fashion MNIST dataset was kept as-is, having one variation with a different data distribution.

3.2.1. Two Class MNIST

The first dataset used throughout all experiments is derived from the MNIST dataset [10]. The original MNIST consists of 10 classes, one for every digit between 0 and 9; see section F for some samples. In order to reduce complexity, two digits were chosen, and the corresponding samples from those digits were taken. The chosen digits were 4 and 9, as these happened to give the most errors from the classifier of the original work of MNIST.

After these samples were selected, the dataset was made 'federated', meaning that it was split up into ten smaller datasets. For this, three different splits were made: an IID distribution, a distribution with imbalances in sample size (SI) and one with imbalances in classes (CI). figure C.1 shows these sample and class distributions.

3.2.2. Four Class MNIST

Besides the 2class MNIST dataset, a 4class MNIST dataset was constructed as well. This contained the same digits as the two-class (four and nine) as a base. From there, the one digit that added the most amount of errors from the original list of errors mentioned earlier was added. This process was repeated once more to create a four-class dataset.

After this 4class MNIST dataset was created, the same process was used to create three similar federated datasets as was done with the 2 class MNIST, e.g. an IID split, a size imbalance split and a class imbalance split. see figure C.2.

3.2.3. Fashion MNIST

The fashion MNIST dataset [28] was created with the goal in mind to be a stand-in replacement of MNIST with a harder classification task, which would be more suitable to benchmark modern ever-improving classifiers. Similarly to MNIST, it contains 10 classes, consisting of different clothing articles; See appendix G for some samples. Due to its goal of replacing MNIST, its dimensions are exactly that of the original MNIST dataset, i.e. 10.000 test -and 60.000 grayscale images of 28 by 28 pixels.

Unlike MNIST, this dataset was used as is, using all 10 classes. Two different distributions have been used, being an IID distribution as well as a class-imbalanced distribution. Note that this imbalanced distribution is built up differently from the MNIST imbalanced distributions; see appendix figure C.3.

3.2.4. AML dataset (A1-A3)

The final dataset used was taken from a study done by Warnat-Herresthal et al. [26]. In their experiments, they used transcriptomes of gene expressions to predict the presence of acute myeloid leukaemia (AML). These transcriptomes were taken from human peripheral blood mononuclear cells (PBMC) from three different sources, all with their own data generation method; Dataset 1 and 2 used different variations of microarray analysis (MRA) [22], whereas dataset 3 uses RNA-sequencing [14]. Due to this heterogeneity of generation methods, the data was separated in three different sets, based on generation method (A1-3).

All three datasets contain 12709 different gene expression values per sample, with labels for 25 different illnesses. With regards to labelling, the same approach was used as in [26], meaning that all AML samples were given a label (1), and all other labels were joined under one label (0) (in the original paper named 'cases' and 'controls'). This approach does result in an inherent class imbalance in the dataset, with approximately 5000 samples of label 0 and only 2500 samples of label 1.

3.3. federated PCA

After some initial testing, it turned out that the high dimensionality of these datasets was going to become problematic. Therefore, a principal component analysis [1] was made to reduce the feature space to 100 features. This PCA was done in a federated way, which is described in algorithm 3 .

3.4. experimental setup

The goal is to compare federated classifier performance to its 'classical', centralized performance. In order to make this comparison fair, certain parameters were set the same. see table 3.2 for the details.

Table 3.2: The equivalent federated parameters to the original centralized ones. Parameters in the same row were kept the same throughout each comparison

centralized	federated
Learning rate	Learning rate (at every client)
epochs	global rounds
batch size	batch size

All experiments were executed on one laptop, running all clients as well as the server. In order to ensure full independence between all clients, the federated learning platform vantage6 [17] [15]was used. Vantage6 uses a dockerized solution, which means that every client (and every task that every node runs on) runs in its own docker, therefore being unable to alter the solution of the other clients.

Algorithm 3 Pseudocode describing the federated PCA process

```
On Server: request metadata from all clients i
On all Clients: $mean^i = mean(X^i)$, $std^i = std(X^i)$, $S^i = length(X^i)$
On server: collect $mean^i$, $std^i$ and $S^i$ for all i
On server: $S = \sum_{i=1}^{n} s^i$
On server: $mean^g = \frac{1}{S} \sum_{i=1}^{n} s^i * mean^i$
On server: send $mean^g$ to all clients, request local covariance matrices
On all clients: $X_n^i = X^i - mean^g$
On all clients: $A^i = (X_n^i)^T * X_n^i$
On server: collect $A_i$ for all i
On server: $A^g = \frac{\sum_{i=1}^{n} A^i}{(\sum_{i=1}^{n} s^i)-1}$
On server: $std^g = diag(A^g)$
On server: send $mean_g, std_g$ to all clients, request local covariance matrices
On all Clients: $X_{n*}^i = \frac{X_i - mean_g}{std_g}$
On all Clients: $A_n^{i*} = (X_{n*}^i)^T * X_{n*}^i$
On server: collect $A_n^{i*}$ for all i
On server: $A_{n*}^g = \frac{\sum_{i=1}^{n} A_{n*}^i}{(\sum_{i=1}^{n} s_i)-1}$
On server: V = eigenvectors($A_n^g$, 100)        ▷ Calculate the first 100 eigenvectors of $A_n^g$
On server: send V to all clients i
On all clients: $X_{PCA}^i = X^i * V$
On all clients: save $X_{PCA}^i$, send 'done' back to server
```

4

Results

Any combination of classifier and dataset which are described below has been made, and experimented on. For sake of brevity, this section will only include some results. A full overview of all results can be found in appendix E.

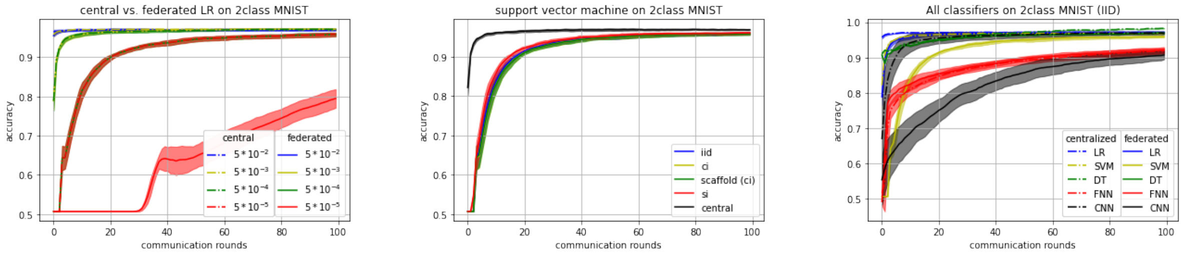
Linear models on binary classification problem show importance of learning rate for convergence time

First, a baseline was created by training a 'simple' classifier on a 'simple' dataset. From there, extensions to both different datasets and classifiers were made. The MNIST dataset, consisting of 60.000 handwritten images of the numbers zero to nine, was chosen as a first dataset, as it is known to be a relatively easy problem for modern day classifiers [28]. In order to simplify further, MNIST was converted into a binary classification problem by selecting only two of the classes. This dataset will be referred to as the 2class MNIST dataset. It was distributed evenly among ten clients, meaning that each client had a similar amount of samples, with an even class distribution.

As for the first classifier, a logistic regression (LR) model (details in methods) was trained on this 2class MNIST dataset, with varying learning rates for both the central and federated model. Results can be found in figure 4.1a, with the final accuracy's and Area Under Curve (AUC) values collected in table D.1. These results show that increasing the learning rate leads to faster convergence times, both in the federated and centralized case. Curiously, a centralized model with a learning rate of a factor 10 lower seems to fit the federated counterpart much better compared to a central model with a learning rate equal to that of a federated system.

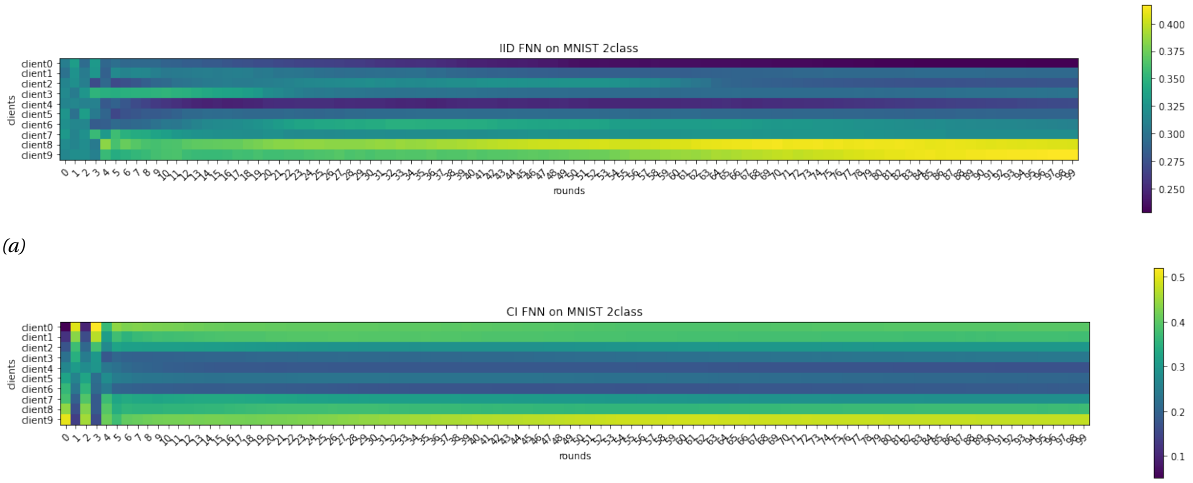
Binary classification experiments show robustness to distribution perturbations
In the next experiment, the robustness to a change in the class -and sample distribution (among the clients) was tested. Previously, 2class MNIST was distributed evenly over all ten clients (referred to as the IID distribution). Now, the same dataset was distributed twice more, but the distribution of either sample size or classes was made differently (methods). These two new distributions are referred to as the sample imbalanced (SI) and class imbalanced (CI) distributions (figure C.1). Furthermore, a Support Vector Machine (SVM) was used besides the already existing LR model (details in methods). Figure 4.1b / table D.2 shows that all federated models give a similar performance compared to their IID counterpart.

Besides fedAVG, another federated algorithm was implemented to explore difference in performance in the CI case. This algorithm is called SCAFFOLD [6] (section 2.2.2). Its performance can be found in figure 4.1b, and is comparable to fedAVG in this setting.



(a) *(b)* *(c)*

Figure 4.1: Several experiments on the 2class MNIST dataset. Graphs display the average and variance over 4 runs. (a) explores the effect of altering the learning rate for both a central and federated logistic regressor(numbers in legend correspond to learning rate). (b) shows the effect of different data distributions (imbalance in dataset size (SI), or in class distribution (CI)) on the SVM. (c) gives an overview of all classifiers, with IID data distributions.



(a)

(b)

Figure 4.2: Figures show heatmaps illustrating the differences between individual client's model updates

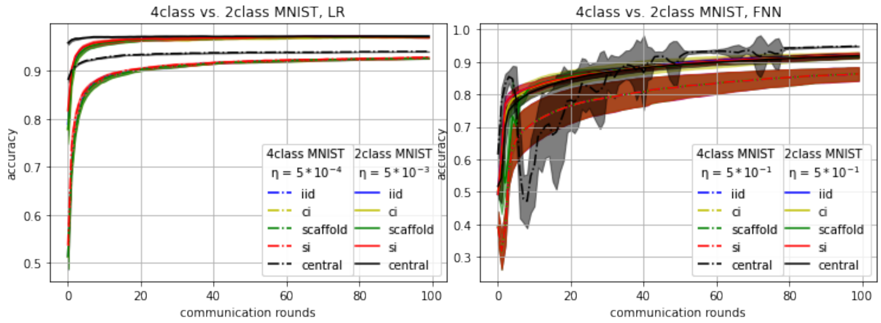
To visualize the learning process, the euclidean distance between each client's parameters and the global parameters were calculated each round. These distances were plotted into a heatmap (figures 4.2a and 4.2b). Figure 4.2b shows an interesting oscillation, which can be explained by the class imbalance on that experiment. The initial model might include a bias towards one of the classes, as it is randomly initialized. This means that the client where the favored class is over-represented will create an update with a smaller magnitude, as its error will be lower. On the other hand, the client on which this favored class is under-represented will create a larger model update. This is seen in the heatmap, as client 0 has a small distance to the global model, whereas client 9 a very large one, with this distance increasing as we move from client 0 to 9. As a result, The next global model iteration might swing more towards the other class (in this case the one over-represented on client 9), which now produces an update with small magnitude, while client 0 produces a much larger update. This cycle continues while the model converges towards an equilibrium, in which both classes are equally represented in the global model. This does mean that both client 0 and 9 create larger model updates compared to the other clients, as is expected.

Extension towards more complex models shows similar results compared to linear classifiers

Once the experiments using the linear models were finished, additional classifiers were tested to explore the effects of increasing model complexity. These were a Fully connected Neural Net (FNN), Convolutional Neural Net (CNN) and a decision tree based classifier (details in methods). Results of all classifiers on the 2class MNIST can be found in figure 4.1c, which shows similar results for both the FNN and decision tree classifiers compared to the linear models previously described. Numerical values for AUC and accuracy at the final training round are found in table D.3. The CNN, however, shows a slightly larger accuracy difference between its central and federated models.

Extension to multiclass problem results in similar performance compared to binary classification
Next, the effects of having a multiclass problem were explored by making an extension to the 2class MNIST dataset. This was done by selecting two more classes from the original MNIST dataset which were then added to the preexisting 2class MNIST dataset, resulting in the 4class MNIST dataset. The results in figure 4.3 / table D.4 show similar results compared to the binary classification problem of 2class MNIST.

The aforementioned 4class MNIST dataset was also distributed in three different ways, similar to the 2class MNIST dataset. Although the accuracy difference compared to the 2class MNIST was significant, the same robustness to distribution perturbation as observed in the 2class MNIST experiments is observed here as well.

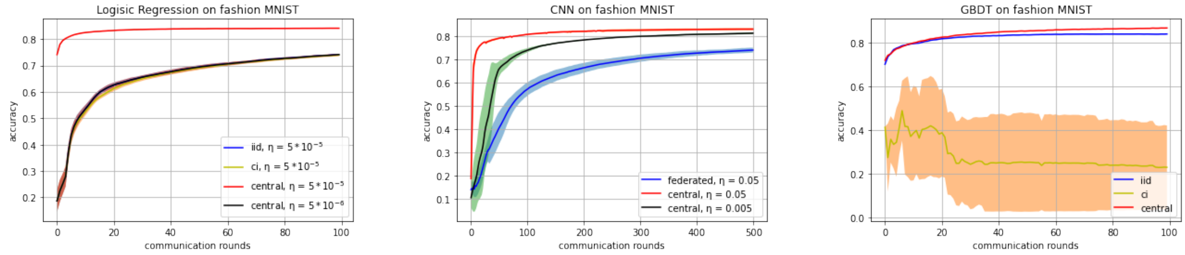


(a) *(b)*

Figure 4.3: Both LR and FNN results on the 4class MNIST, compared to previously displayed 2class MNIST results.

Increasing class distribution aggresiveness gives variable results
After creating working models on the MNIST derivatives, the next goal was to experiment on a 'harder' dataset. Here the fashion MNIST dataset was used , as it is meant to be a direct replacement and at the same time more difficult problem compared to the original MNIST [28].

Two different dataset distributions were made, one with an IID distribution, and one with a class-imbalanced (CI) distribution. This time, however the CI distribution was made more 'aggressive', meaning that not every client had samples for every class (figure C.3).
Results on fashion MNIST can be found in figure 4.4 / table D.5. The linear models kept performing similarly, whereas the neural nets show a significant accuracy difference between the federated and centralized models. Difference between IID and CI distributions seem negligible, except for the GBDT, which completely breaks down in the CI case.



(a) *(b)* *(c)*

Figure 4.4: Several models on the fashion MNIST dataset. (a) underlines the increased convergence time (note the x axis), while (b) and (c) compare with the different distributions.

Experiments on highly dimensional data show potential for federated data processing techniques
Until now, all experiments have utilized a single dataset which has then been separated into multiple pieces. A more realistic scenario would use multiple datasets, where each dataset is held on a separate client.

In order to imitate this scenario more realistically, a triad of datasets (A1-A3) from [26] was used. these three datasets are representations of gene expressions, labelled based on the presence or absence of Acute Myeloid Leukemia (AML) in a sample (each sample corresponds to blood cells of a different human). These three datasets were synthesized independently, with A3 even having a different measuring method compared to A1/A2; A1-2 uses microarray analysis, whereas A3 uses RNA-sequencing (methods). This dataset will be referred to as the AML dataset from now on.

In order to determine the feasibility of learning on these datasets, first an experiment was done using A2 only. Here, A2 was distributed in an IID fashion over 10 clients. As A2 has a high dimensionality (over 11.000 features), a dimensionality reduction method seemed to be required, as all classifiers (also in the centralized models) could not perform above random level in the original feature domain. Therefore, a Principal Component Analysis (PCA) was performed beforehand, reducing to 100 features. This was done in a federated manner, which is described in more detail in the methods.

With the reduced dataset, all classifiers seemed to be performing reasonably well. Once the IID experiment was completed, the data was distributed in a SI/CI distribution similar to the MNIST datasets, before reapplying the federated PCA. Once again a high robustness to these different data distributions can be seen, see figure 4.5 / table D.6.

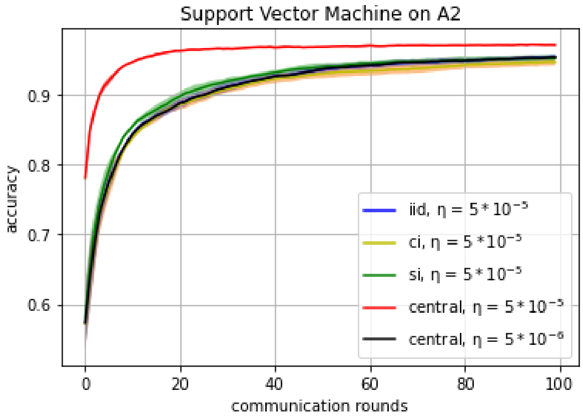


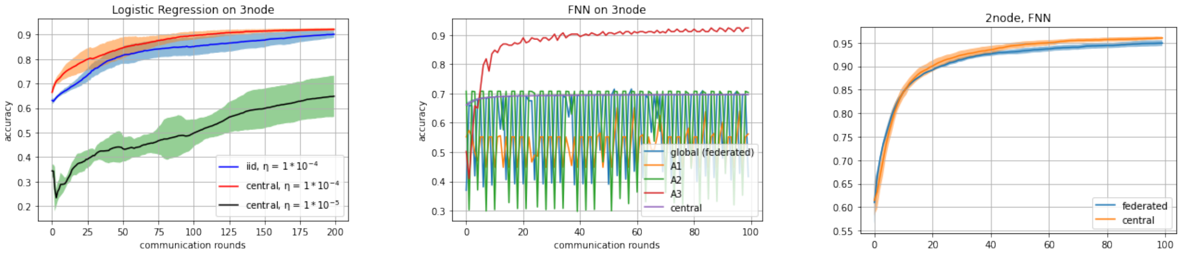
Figure 4.5: SVM results on the A2 dataset

Usage of heterogeneous datasets can inhibit proper model training
Once experiments on only the A2 dataset were complete, the full triad of A1-A3 was tested. In this setup, only three clients have been used, each holding one complete dataset. As these datasets have the same high dimensonality issue as the A2 dataset described before, first the federated PCA was executed.

The results vary per type of classifier; Whereas the linear models seem to be able to perform reasonably well (figure 4.6a), the neural networks completely break down, both in the central case as well as the federated case (see figure 4.6b).

The hypothesis on why this breakdown is happening was that one of these datasets, A3, is distributed quite differently in such a way that it disturbs training. The reason to select A3 specifically is that it has been synthesized using a different measuring technique (methods); besides, the preprocessing steps of the original authors, which were followed in this experiment, are also different for A3. Furthermore, when looking at a heatmap of the test (figure 4.7), it can be seen that the model updates from the client holding A3 are consistently far away from the global average model.

In order to test this hypothesis, an experiment was conducted in which the A3 dataset was omitted; In this case there were only two clients, holding A1 and A2 respectively. A new PCA was made using only these two clients. Results show not only that the central and federated neural models are quite similar in performance, but their performance also ends up being quite agreeable, indicating that the use of A3 indeed disturbed the convergence of training (figure 4.6c).



(a) *(b)* *(c)*

Figure 4.6: Several experiments on the A1-A3 datasets. figure 4.6c shows an improvement over 4.6b by omitting dataset A3.

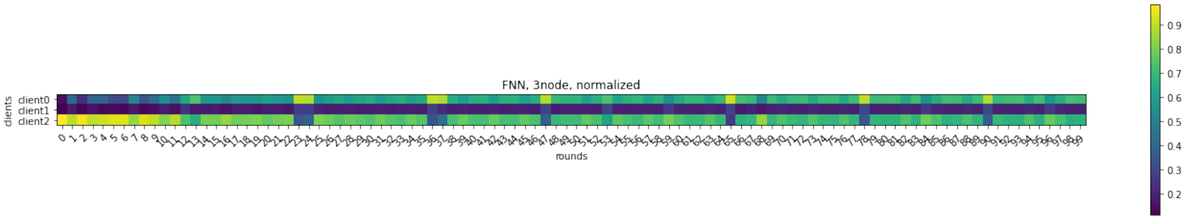


Figure 4.7: The heatmap of the FNN on a 3node dataset experiment, corresponding to figure 4.6b.

5 Conclusion and Discussion

This thesis describes the results of a set of experiments exploring the differences between centralized and federated machine learning models. Multiple classifiers were used on several datasets, which have been distributed in different ways. Results show that, especially under IID circumstances, a federated classifier could reach similar performances compared to its centralized counterpart. However, a careful choice of parameters such as learning rate and batch size is vital to reach a comparable performance. The exact relation between said parameters for reaching equal performance between centralized and federated models remains unclear, and could be part of future work. Furthermore, when using datasets from multiple sources, which is a paradigm explicitly supported by the federated setting, the heterogeneity between said sources could arise, and should be approached catiously.

On the MNIST 2class dataset, the federated convolutional neural network scores considerably lower than its central version (fig. 4.1c). Moreover, it is the only classifier with such a big discrepancy. This could be the result of the batch learning used for the CNN classifier; no batch learning has been utilized for any other model. A possible explanation could be that a central model utilizes batch learning more efficiently than a federated model.

Figure 4.1a shows that when comparing a centralized and federated model, a learning rate of a factor 10 lower for the central model gives a much better fit instead of using equal learning rates. The reason for this difference is not entirely clear. However, it is intriguing that this factor of 10 is equal to the amount of clients used in the federated case, making this a potential starting point for future research.

Comparing the results on the MNIST 2class dataset to its extension, the 4class dataset, accuracy differences between the federated and central models seem to increase (fig 4.3). Although the learning rate used in the 4class case is lower, it is kept equal between the central and federated models, and should therefore not be of influence for any final accuracy difference. A possible explanation could be that intuitively the problem becomes 'harder' when moving from 2 to 4 classes, therefore amplifying any performance differences between a centralized and a federated model.

As previously described, the federated GBDT-classifier completely breaks down on the class-imbalanced distribution of the fashion MNIST dataset (figure 4.4c). This is not surprising looking at the dataset distribution, which shows that all clients have approximately half of the total amount of classes. The inPrivate learning algorithm utilizes decision trees created by the previous clients to boost; if these trees were created using a different subset of classes, it can be understood that the resulting gradient does not serve as a helpful tool for the creation of the following decision tree.

In [26], from which the AML dataset originated, no form of dimensionality reduction was used. Furthermore, no omission of dataset A3 in order to improve results was mentioned. The first point can be explained by the difference in the used classifiers. In this work, a deep neural network including multiple dropout layers, which are known for being helpful at dealing with highly dimensional data, was used. In contrast, the neural network used here only consisted of two linear layers, combined with a relu-layer. Regarding the use of A3, it is unclear of the original results were synthesized using federated averaging; The authors state a couple of other aggregation methods that were used, including taking the median instead. This, combined with the aforementioned point on model difference, could lead to a setup in which A3 does not bring much of a disturbance.

One of the strengths of the federated averaging algorithm is that it allows for multiple local epochs, as well as batch learning. This results in multiple learning steps per communication round. In the original work [16], the authors show that this can be used to reduce the total amount of communication rounds required. In this work, neither local epochs nor batch learning has been used (except for the CNN, where 10 batches per round were used. However, this number was never varied on to explore its influence). A further analysis on differences between federated and central models should probably include experiments exploring the influence of varying these parameters.

Bibliography

[1] Michael Galarnyk. "PCA using Python (scikit-learn)". In: *towards data science* (2017). URL: https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60.

[2] Jonas Geiping et al. "Inverting Gradients - How easy is it to break privacy in federated learning?" In: *CoRR* abs/2003.14053 (2020). arXiv: 2003.14053. URL: https://arxiv.org/abs/2003.14053.

[3] Farzin Haddadpour and Mehrdad Mahdavi. "On the Convergence of Local Descent Methods in Federated Learning". In: *CoRR* abs/1910.14425 (2019). arXiv: 1910.14425. URL: http://arxiv.org/abs/1910.14425.

[4] Eunjeong Jeong et al. "Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data". In: *CoRR* abs/1811.11479 (2018). arXiv: 1811.11479. URL: http://arxiv.org/abs/1811.11479.

[5] Arthur Jochems et al. "Distributed learning: Developing a predictive model based on data from multiple hospitals without data leaving the hospital – A real life proof of concept". In: *Radiotherapy and Oncology* 121.3 (2016), pp. 459–467. ISSN: 0167-8140. DOI: https://doi.org/10.1016/j.radonc.2016.10.002. URL: https://www.sciencedirect.com/science/article/pii/S0167814016343365.

[6] Sai Praneeth Karimireddy et al. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 5132–5143. URL: https://proceedings.mlr.press/v119/karimireddy20a.html.

[7] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtarik. "Tighter Theory for Local SGD on Identical and Heterogeneous Data". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 4519–4529. URL: https://proceedings.mlr.press/v108/bayoumi20a.html.

[8] Kiprono Elijah Koech. "Cross-Entropy Loss Function". In: *towards data science* (2020). URL: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[9] Jakub Konečný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *CoRR* abs/1610.05492 (2016). arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492.

[10] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[11] Qinbin Li et al. "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection". In: *CoRR* abs/1907.09693 (2019). arXiv: 1907.09693. URL: http://arxiv.org/abs/1907.09693.

[12] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749.

[13] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749.

[14] Ruairi J Mackenzie. "RNA-Seq: Basics, Applications and Protocol". In: *Technology Networks* (2017). URL: https://www.technologynetworks.com/genomics/articles/rna-seq-basics-applications-and-protocol-299461.

[15] Frank Martin, Melle Sieswerda, and et al. Hasan Alradhi. *vantage6 repository*. Accessed: 2021-09-08. URL: https://doi.org/10.5281/zenodo.3686944.

[16] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[17] Arturo Moncada-Torres et al. "VANTAGE6: an open source priVAcy preserviNg federaTed leArninG infrastructurE for Secure Insight eXchange". In: *AMIA Annual Symposium Proceedings*. 2020, pp. 870–877.

[18] Angelia Nedic and Asuman Ozdaglar. "Distributed Subgradient Methods for Multi-Agent Optimization". In: *IEEE Transactions on Automatic Control* 54.1 (2009), pp. 48–61. DOI: 10.1109/TAC.2008.2009515.

[19] Angelia Nedić et al. "Geometrically convergent distributed optimization with uncoordinated step-sizes". In: *2017 American Control Conference (ACC)*. 2017, pp. 3950–3955. DOI: 10.23919/ACC.2017.7963560.

[20] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Guannan Qu and Na Li. "Harnessing Smoothness to Accelerate Distributed Optimization". In: *IEEE Transactions on Control of Network Systems* 5.3 (2018), pp. 1245–1260. DOI: 10.1109/TCNS.2017.2698261.

[22] John Quackenbush. "Microarray Analysis and Tumor Classification". In: *New England Journal of Medicine* 354.23 (2006). PMID: 16760446, pp. 2463–2472. DOI: 10.1056/NEJMra042342. eprint: https://doi.org/10.1056/NEJMra042342. URL: https://doi.org/10.1056/NEJMra042342.

[23] Anit Kumar Sahu et al. "On the Convergence of Federated Optimization in Heterogeneous Networks". In: *CoRR* abs/1812.06127 (2018). arXiv: 1812.06127. URL: http://arxiv.org/abs/1812.06127.

[24] Wei Shi et al. "EXTRA: An Exact First-Order Algorithm for Decentralized Consensus Optimization". In: *SIAM Journal on Optimization* 25.2 (2015), pp. 944–966. DOI: 10.1137/14096668X. eprint: https://doi.org/10.1137/14096668X. URL: https://doi.org/10.1137/14096668X.

[25] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. "Membership Inference Attacks against Machine Learning Models". In: *CoRR* abs/1610.05820 (2016). arXiv: 1610.05820. URL: http://arxiv.org/abs/1610.05820.

[26] Stefanie Warnat-Herresthal et al. "Swarm Learning for decentralized and confidential clinical machine learning". In: *Nature* 594.7862 (June 2021), pp. 265–270. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03583-3. URL: https://doi.org/10.1038/s41586-021-03583-3.

[27] Kang Wei et al. "Federated Learning With Differential Privacy: Algorithms and Performance Analysis". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469. DOI: 10.1109/TIFS.2020.2988575.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[29] Jie Xu et al. "Federated Learning for Healthcare Informatics". In: *Journal of Healthcare Informatics Research* 5.1 (Mar. 2021), pp. 1–19. ISSN: 2509-498X. DOI: 10.1007/s41666-020-00082-4. URL: https://doi.org/10.1007/s41666-020-00082-4.

[30] Tao Yang et al. "A survey of distributed optimization". In: *Annual Reviews in Control* 47 (2019), pp. 278–305. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2019.05.006. URL: https://www.sciencedirect.com/science/article/pii/S1367578819300082.

[31] Lingchen Zhao et al. "InPrivate Digging: Enabling Tree-based Distributed Data Mining with Differential Privacy". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 2087–2095. DOI: 10.1109/INFOCOM.2018.8486352.

A

federated averaging derivation

First, a derivation is given considering a singular local epoch per communication round, as well as no batch learning. This is then afterwards extended including both the effect of multiple local epochs as well as batch learning.

A.1. No batch learning or local epochs

The local model update from federated averaging is given by stochastic gradient descent:

$$W_t^i = W_{t-1}^g - \eta \nabla L(W_{t-1}^g(X^i), y^i) \tag{A.1}$$

These local model updates are then combined into a new global model as follows:

$$W_t^g = \frac{1}{S}\sum_{i=1}^{N} s^i W_t^i \tag{A.2}$$

Where N is the amount of clients, s^i the dataset size of client i, and $S = \sum_{i=1}^{N} s^i$. If we combine equations A.1 and A.2, we get:

$$W_t^g = \frac{1}{S}\sum_{i=1}^{n}\{s^i[W_{t-1}^g - \eta \nabla L(W_{t-1}^g(X^i, y^i)]\} = \frac{1}{S}\sum_{i=1}^{n} s^i[W_{t-1}^g - f^i(t-1)], \tag{A.3}$$

with $f^i(t) = \eta \nabla L(W_t^g(X^i), y_t^i)$. By doing some refactoring, equation A.3 becomes:

$$W_t^g = \frac{1}{S}\sum_{i=1}^{n} s^i W_{t-1}^g - \frac{1}{S}\sum_{i=0}^{n} s^i f^i(t-1) = \frac{W_{t-1}^g}{S}\sum_{i=1}^{n} s^i - \frac{1}{S}\sum_{i=1}^{n} s^i f^i(t-1). \tag{A.4}$$

Since $S = \sum_{i=1}^{n} s^i$, we get:

$$W_t^g = W_{t-1}^g - \frac{1}{S}\sum_{i=0}^{n} s^i f^i(t-1). \tag{A.5}$$

This recursion can be rewritten towards its initial condition, being:

$$W_T^g = W_0^g - \sum_{t=1}^{T}\frac{1}{S}\sum_{i=1}^{n} s^i f^i(t), \tag{A.6}$$

Where W_0^g is the random initialization of the model. Equation A.6 can be generalized even further, if we make two important assumptions. The first is that all client data is equally distributed, i.e. **the data distribution amongst clients is IID**. The second assumption, is that, if the first assumption holds, **each client delivers a similar model update**. In this case, we get that $f^i(t) = f(t)$, and equation A.6 becomes:

$$W_T^g = W_0^g - \sum_{t=1}^{T}\frac{1}{S}\sum_{i=1}^{n} s^i f(t) = W_0^g - \sum_{t=1}^{T} f(t) = W_0^g - \sum_{t=1}^{T}\eta \nabla L(W_{t-1}^g(X, y)), \tag{A.7}$$

Which is equivalent to Stochastic Gradient Descent in a centralized model.

A.2. Generalization using batch learning and local epochs

First, if we introduce batch learning (only), equation A.1 becomes:

$$W_t^i = W_{t-1}^g - \sum_{b=1}^{B} \eta \nabla L(W_{t-1,b}^g(X_b^i), y_b^i), \tag{A.8}$$

Where B is the total amount of batches. Similarly, using only local epochs (and no batch learning), we can get:

$$W_t^i = W_{t-1}^g - \sum_{e=1}^{E} \eta \nabla L(W_{t-1,e}^g(X^i), y^i), \tag{A.9}$$

with E the total amount of local epochs. Combining equations A.8 and A.9, we get:

$$W_t^i = W_{t-1}^g - \sum_{e=1}^{E} \sum_{b=1}^{B} \eta \nabla L(W_{t-1,b,e}^g(X_b^i), y_b^i). \tag{A.10}$$

Note that equation A.2 is independent of either batch learning or local epoch amount. Following the same derivation as in the previous section, but with equation A.10 instead of A.1, we can arrive at:

$$W_t^g = W_{t-1}^g - \frac{1}{S} \sum_{i=1}^{n} s^i \sum_{e=1}^{E} \sum_{b=1}^{B} f^i(t-1, b, e), \tag{A.11}$$

where $f^i(t-1,b,e) = \eta \nabla L(W_{t-1,b,e}^g(X_b^i), y_b^i)$. Once again, if we make the same two assumptions as in the previous section (data is distributed IID amongst clients, each client update results in a similar gradient over the loss function), we can generalize using $f^i(t,b,e) = f(t,b,e)$:

$$W_t^g = W_{t-1}^g - \frac{1}{S} \sum_{i=1}^{n} s^i \sum_{e=1}^{E} \sum_{b=1}^{B} f(t-1,b,e) = W_{t-1}^g - \sum_{e=1}^{E} \sum_{b=1}^{B} f(t-1,b,e) = W_0^g - \sum_{t=1}^{T} \sum_{e=1}^{E} \sum_{b=1}^{B} f(t-1,b,e) \tag{A.12}$$

B
Classifiers

Logistic Regression (LR)
Logistic Regression is a linear model, meaning that it tries to learn a linear function given by

$$f(x) = w^T x + b \tag{B.1}$$

where x is a set of training samples, and w^t and b the coefficients to be learned. This is done by minimizing a loss function; The difference between these loss functions is what differentiates between the different linear classifiers in our experiments. for the Logistic regressor, the loss function is given by [20]:

$$L_{LR}(y_i, f(x_i) = log(1 + exp(-y_i f(x_i))) \tag{B.2}$$

where y_i is the label of training sample x_i.

Support Vector Machine (SVM)
The Support Vector Machine is also a linear model, so the function it tries to learn follows eq. B.1, just like the logistic regressor. The difference with LR lies in the loss function, which for SVM notes [20]:

$$L_{SVM}(y_i, f(x_i)) = max(0, 1 - y_i f(x_i)) \tag{B.3}$$

fully connected neural network (FNN)
The FNN architectures have been kept as simple as possible, as attaining maximal accuracy was not a goal of the experiments. see table B.1 for the details. Note that the only perturbations between the different datasets is within the final layer, to accommodate for the appropriate amount of classes. The loss function used for updating the coefficients is the cross-entropy loss [8] :

$$L(x_i, y_i) = -\sum_{i=1}^{c} y_i log(x_i) \tag{B.4}$$

with c the amount of classes, y_i the truth label (either 0 or 1) of a certain sample and x_i the probability for class i estimated by the neural net.

Table B.1: the architectures of the FNN

	Layer 1	Layer 2	Layer 3
MNIST 2 Class	Fully Connected: 784 x 100	ReLU	Fully Connected: 100 x 2
MNIST 4 Class	Fully Connected: 784 x 100	ReLU	Fully Connected: 100 x 4
fashion MNIST	Fully Connected: 784 x 100	ReLU	Fully Connected: 100x10
A2	Fully Connected: 100 x 100	ReLU	Fully Connected: 100x2

Convolutional Neural Network (CNN)

The CNN architectures have been derived from the FNN architectures; see table B.2. Besides the first layer being changed to a convolutional one, the last fully connected layer has its input dimensions changed in order to fit the output of the convolutional layer. The same loss function from the FNN, i.e. cross-entropy loss (eq. B.4), is used for the CNN implementations as well.

Table B.2: the CNN architectures

	Layer 1	Layer 2	Layer 3

MNIST
2 Class
	Convolutional:
	kernel size:3
	stride: 1
	padding: 1
		Max pool:
		kernel = 2
		stride = 2
			Fully Connected:
			196 x 2

MNIST
4 Class
	Convolutional:
	kernel size:3
	stride: 1
	padding: 1
		Max pool:
		kernel = 2
		stride = 2
			Fully Connected:
			196 x 4

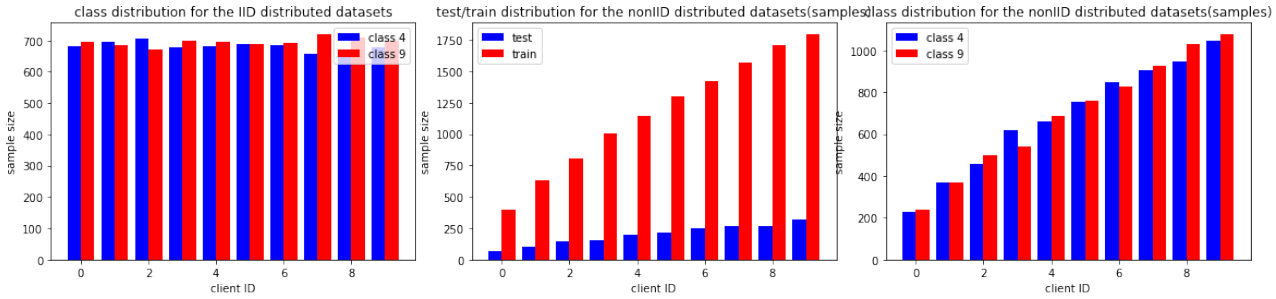
fashion
MNIST
	Convolutional:
	kernel size:3
	stride: 1
	padding: 1
		Max pool:
		kernel = 2
		stride = 2
			Fully Connected:
			196 x 10

A2	Convolutional:
	kernel size:3
	stride: 1
	padding: 1
		Max pool:
		kernel = 2
		stride = 2
			Fully Connected:
			25 x 2

Gradient Boosting Decision Trees (GBDT)

Gradient Boosting Decision Trees is an ensemble classifier that builds a collection of decision trees in a sequential manner. The single decision trees are called 'weak learners', and the concept of gradient boosting can be applied to a variety of weak learners. After a tree has been made, the loss on the training dataset is calculated based on some differentiable loss function. Then, an update step is being performed which is akin to gradient descent. However, instead of modifying the parameters of the first tree, we add another tree which is parameterized such as to lower the loss which was calculated using the first tree. This process then repeats, using the existing trees to calculate a loss, which is then used to parameterize the next tree. Prediction (and thus loss calculation) is performed using a majority vote, weighted by the accuracy of the corresponding tree's iteration.

C
supplementary figures

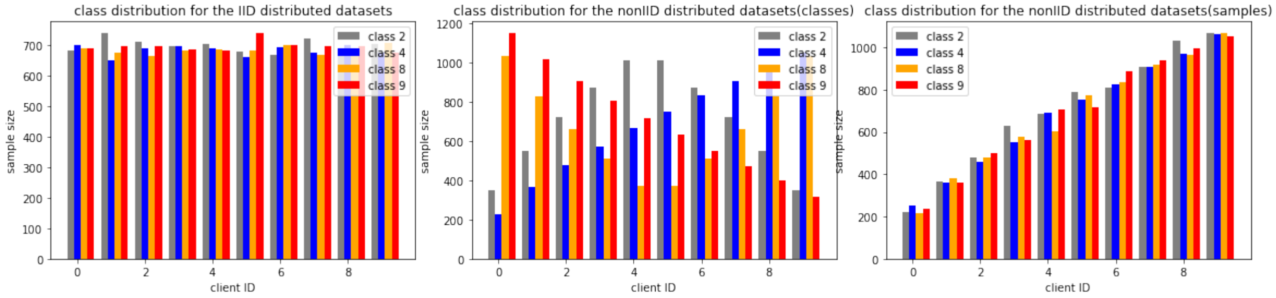


(a) iid

(b) ci

(c) si

Figure C.1: The three different data distributions for the 2class MNIST dataset

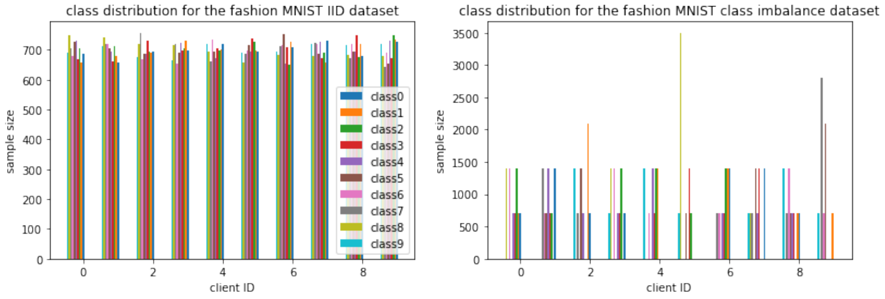


(a) iid

(b) ci

(c) si

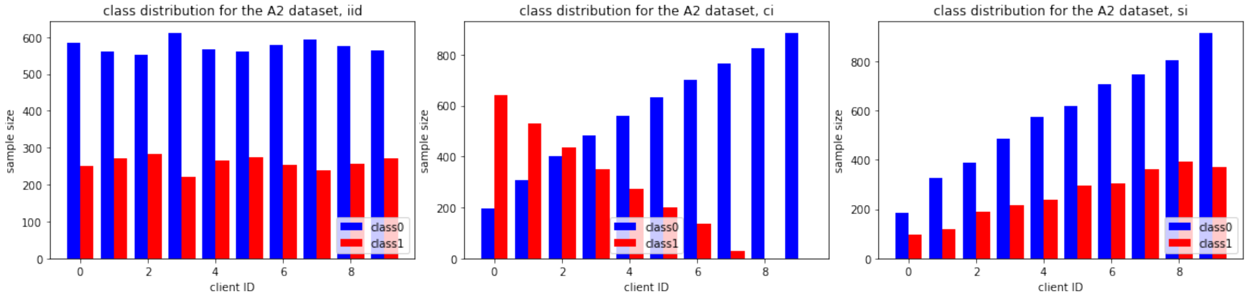
Figure C.2: The three different data distributions for the 4class MNIST dataset



(a)

(b)

Figure C.3: The different data distributions for the fashion MNIST dataset



(a) iid

(b) ci

(c) si

Figure C.4: The three different data distributions for the A2 dataset

D

supplementary tables

Table D.1: Logistic regression results on the 2class MNIST dataset with multiple different learning rates (figure 4.1a). Measured were Area Under Curve (AUC) and the accuracy in the final communication round, both for a centralized and a federated model with equal parameters.

	AUC		Final Accuracy	
Learning rates	Federated	Central	Federated	Central
$5 * 10^{-2}$	0.960	0.959	0.971	0.969
$5 * 10^{-3}$	0.956	0.961	0.971	0.971
$5 * 10^{-4}$	0.899	0.955	0.958	0.971
$5 * 10^{-5}$	0.630	0.899	0.795	0.958

Table D.2: Results of SVM on the 2class MNIST dataset. All experiments used the same learning rate. Data corresponds to figure 4.1b.

	AUC	Accuracy
IID	0.907	0.959
CI	0.905	0.959
SCAFFOLD	0.904	0.958
SI	0.912	0.960
Central	0.955	0.968

Table D.3: Overview of all classifiers on 2class MNIST (IID distribution). Equal learning rate between federated and central model per classifier. Corresponds to figure4.1c

		AUC	Final Accuracy
LR	Federated	0.956	0.965
	Central	0.961	0.970
SVM	federated	0.907	0.911
	Central	0.955	0.965
GBDT	Federated	0.947	0.956
	Central	0.955	0.965
FNN	Federated	0.870	0.891
	Central	0.863	0.876
CNN	Federated	0.815	0.794
	Central	0.946	0.964

Table D.4: Comparison between 2class and 4class MNIST for LR and FNN. learning rates were kept equal per combination of classifier and datasetcorresponds to figure 4.3.

		2class MNIST		4class MNIST	
		AUC	Final Accuracy	AUC	Final Accuracy
LR	IID	0.956	0.971	0.897	0.926
	CI	0.954	0.970	0.897	0.926
	SCAFFOLD	0.955	0.971	0.897	0.926
	SI	0.955	0.970	0.900	0.928
	Central	0.961	0.971	0.926	0.940
FNN	IID	0.870	0.891	0.788	0.806
	CI	0.867	0.851	0.788	00.806
	SCAFFOLD	0.862	0.869	0.788	0.806
	SI	0.870	0.889	0.787	0.806
	Central	0.863	0.876	0.856	0.855

Table D.5: results on the fashion MNIST dataset. corresponds to figure 4.4. federated models were using the learning rate of the upper central option displayed.

		AUC	Final Accuracy
CNN (500 rounds)	IID	0.629	0.740
	Central, $\eta = 5 * 10^{-2}$	0.812	0.830
	Central, $\eta = 5 * 10^{-3}$	0.743	0.813
GBDT	IID	0.817	0.839
	CI	0.275	0.229
	Central	0.833	0.867
LR	IID	0.651	0.780
	CI	0.647	0.740
	Central, $\eta = 5 * 10^{-5}$	0.827	0.842
	Central, $\eta = 5 * 10^{-6}$	0.651	0.742

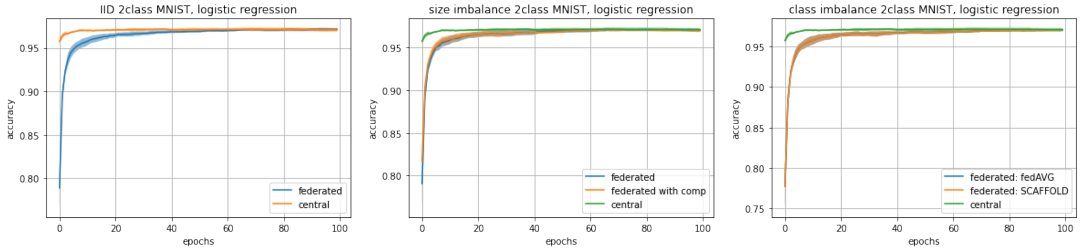
Table D.6: Results of the SVM classifier on the A2 dataset. Corresponds to figure 4.5. All learning rates except for bottom line were equal ($5 * 10^{-5}$)*.*

	AUC	Final Accuracy
IID	0.901	0.954
CI	0.896	0.947
SI	0.907	0.953
Central	0.952	0.974
Central $\eta = 5 * 10^{-6}$	0.901	0.963

E

collection of all results

E.1. 2class MNIST

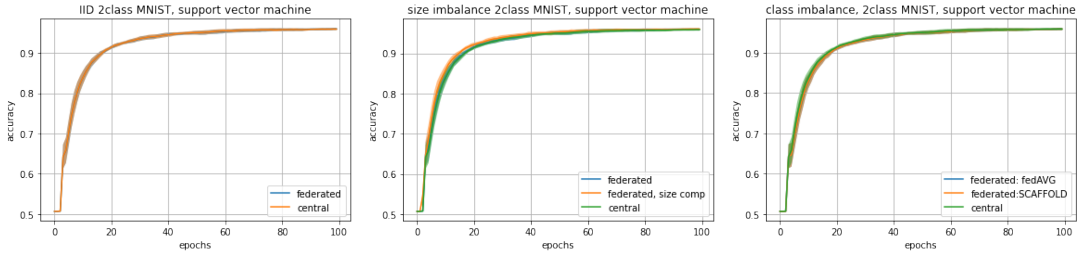


(a) IID

(b) size imbalance

(c) class imbalance

Figure E.1: Logistic regression results on the 2 class MNIST dataset

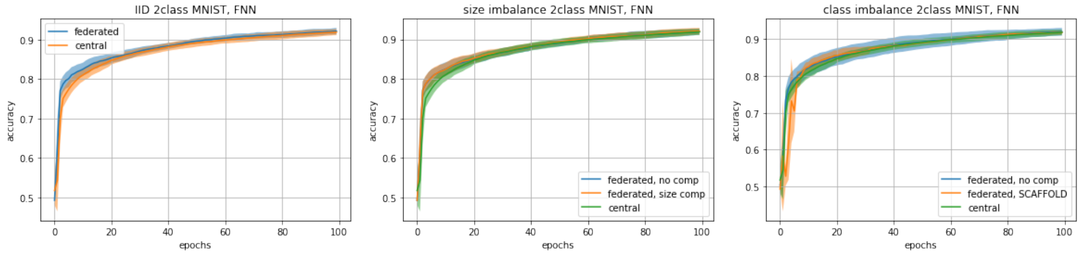


(a) IID

(b) size imbalance

(c) class imbalance

Figure E.2: Support Vector Machine results on the 2 class MNIST dataset

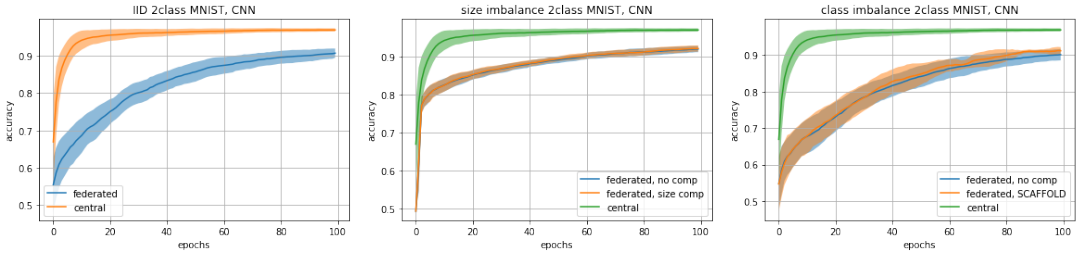


(a) IID

(b) size imbalance

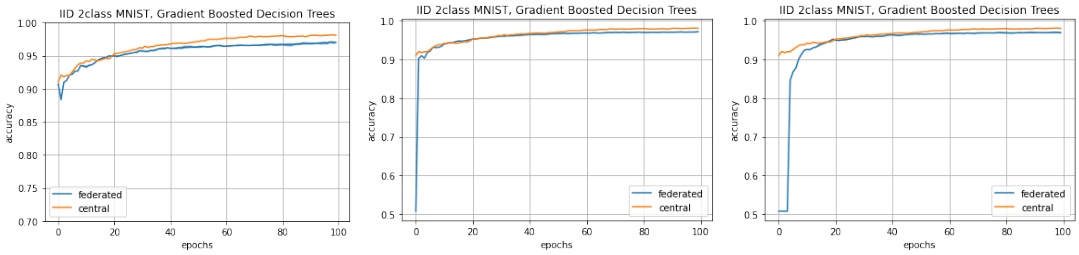
(c) class imbalance

Figure E.3: FNN results on the 2 class MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

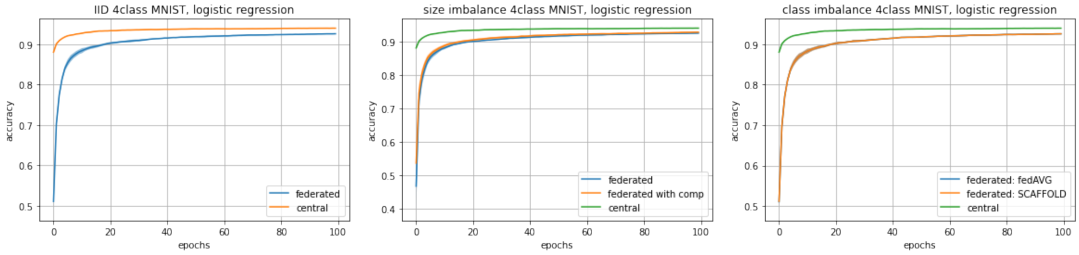
Figure E.4: CNN results on the 2 class MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

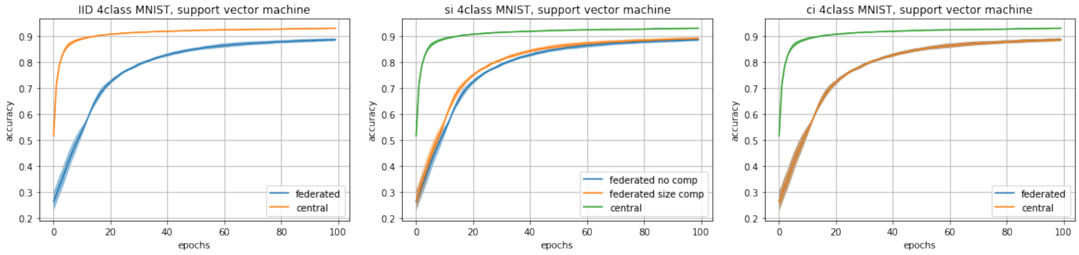
Figure E.5: GBDT results on the 2 class MNIST dataset

E.2. 4class MNIST



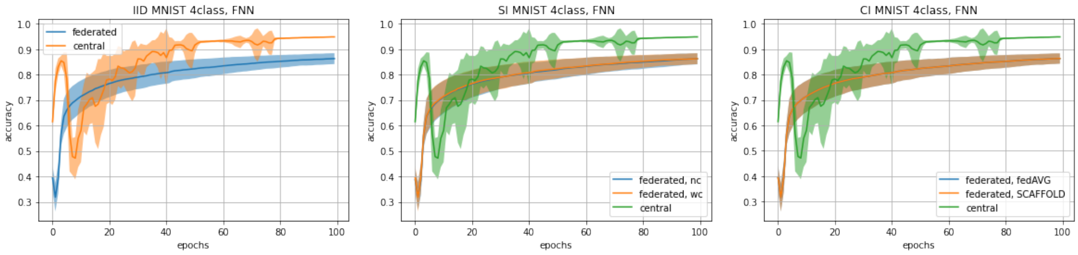
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.6: Logistic regression results on the 4 class MNIST dataset



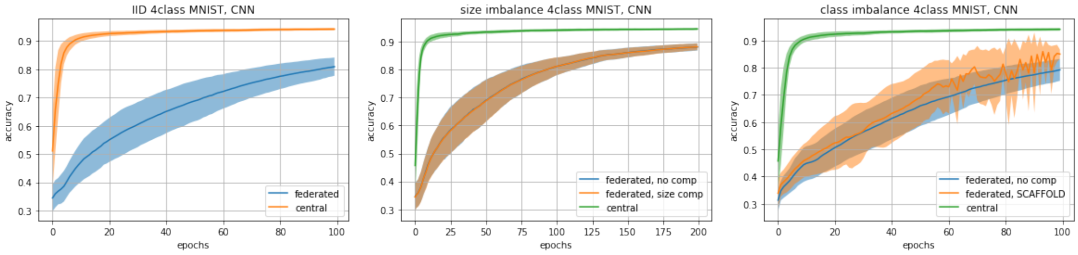
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.7: Support Vector Machine results on the 4 class MNIST dataset



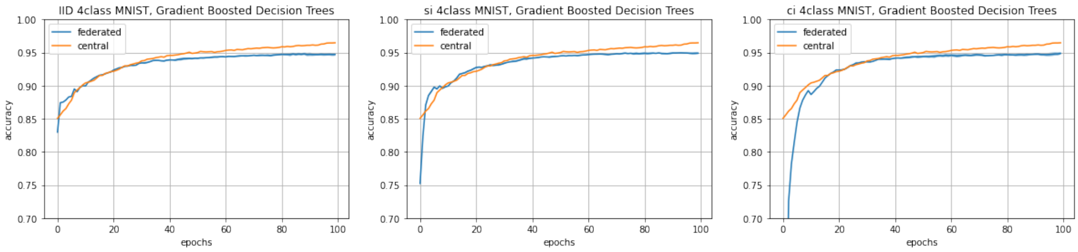
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.8: FNN results on the 4 class MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.9: CNN results on the 4 class MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.10: GBDT results on the 4 class MNIST dataset

E.3. Fashion MNIST

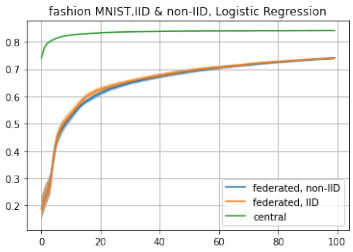


Figure E.11: Logistic regression results on the fashion MNIST datasets

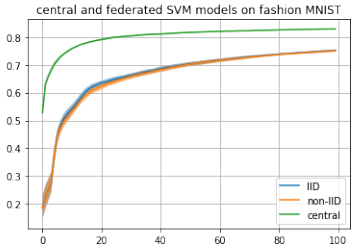


Figure E.12: Support Vector Machine results on the fashion MNIST dataset

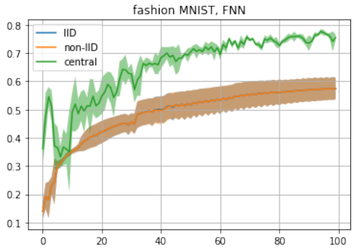


Figure E.13: FNN results on the fashion MNIST dataset

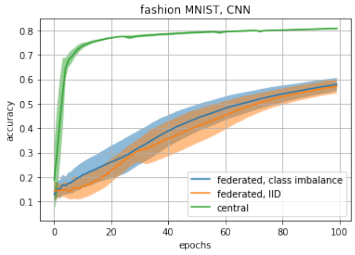


Figure E.14: CNN results on the fashion MNIST dataset

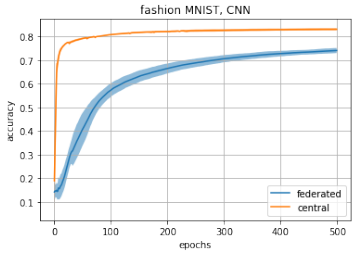


Figure E.15: CNN on fashion MNIST, longer run

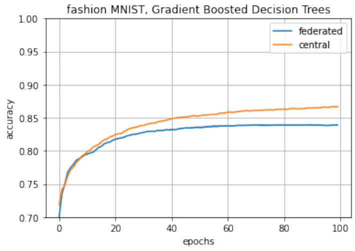


Figure E.16: GBDT results on the fashion MNIST dataset

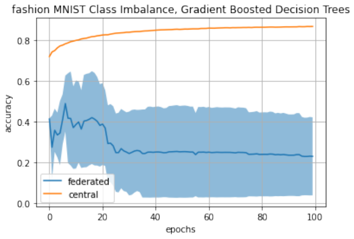
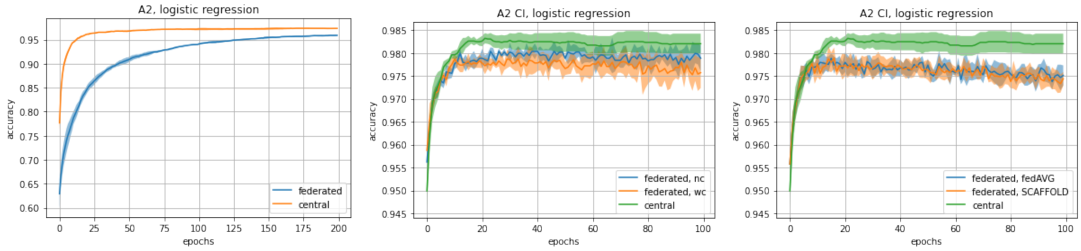


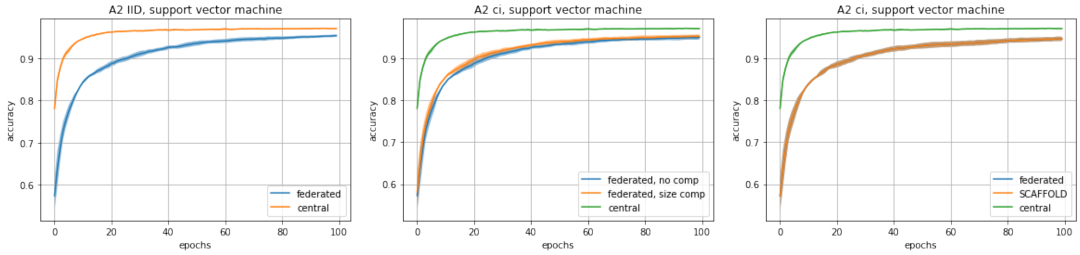
Figure E.17: GBDT results on the fashion MNIST class imbalance dataset

E.4. AML: A2



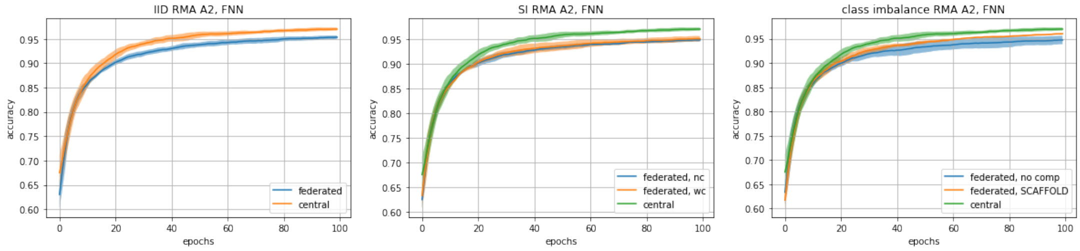
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.18: Logistic regression results on the A2 dataset



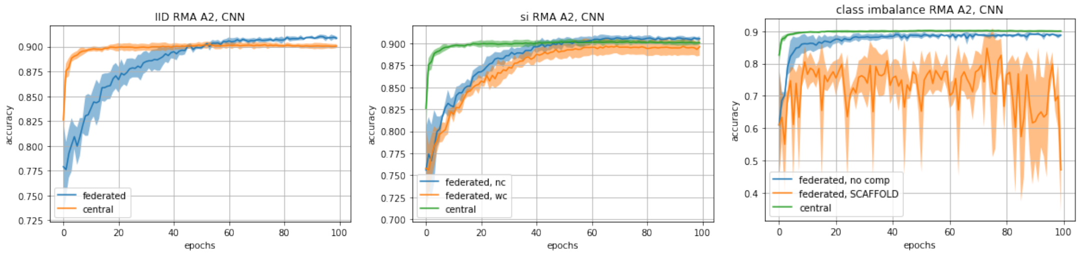
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.19: Support Vector Machine results on the A2 dataset



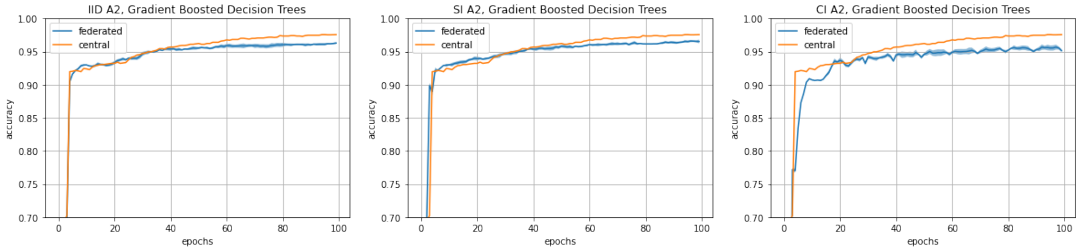
(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.20: FNN results on the A2 MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.21: CNN results on the A2 MNIST dataset



(a) IID *(b) size imbalance* *(c) class imbalance*

Figure E.22: GBDT results on the A2 MNIST dataset

E.5. AML: A1-A3

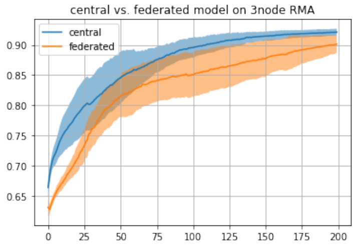


Figure E.23: Logistic regression results on the swarm dataset

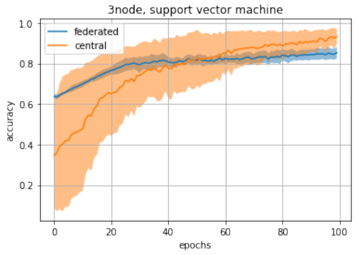


Figure E.24: SVM results on the swarm dataset

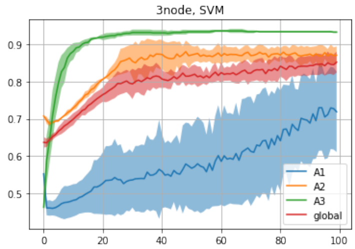


Figure E.25: federated SVM results on the swarm dataset, split out per dataset

F

MNIST samples

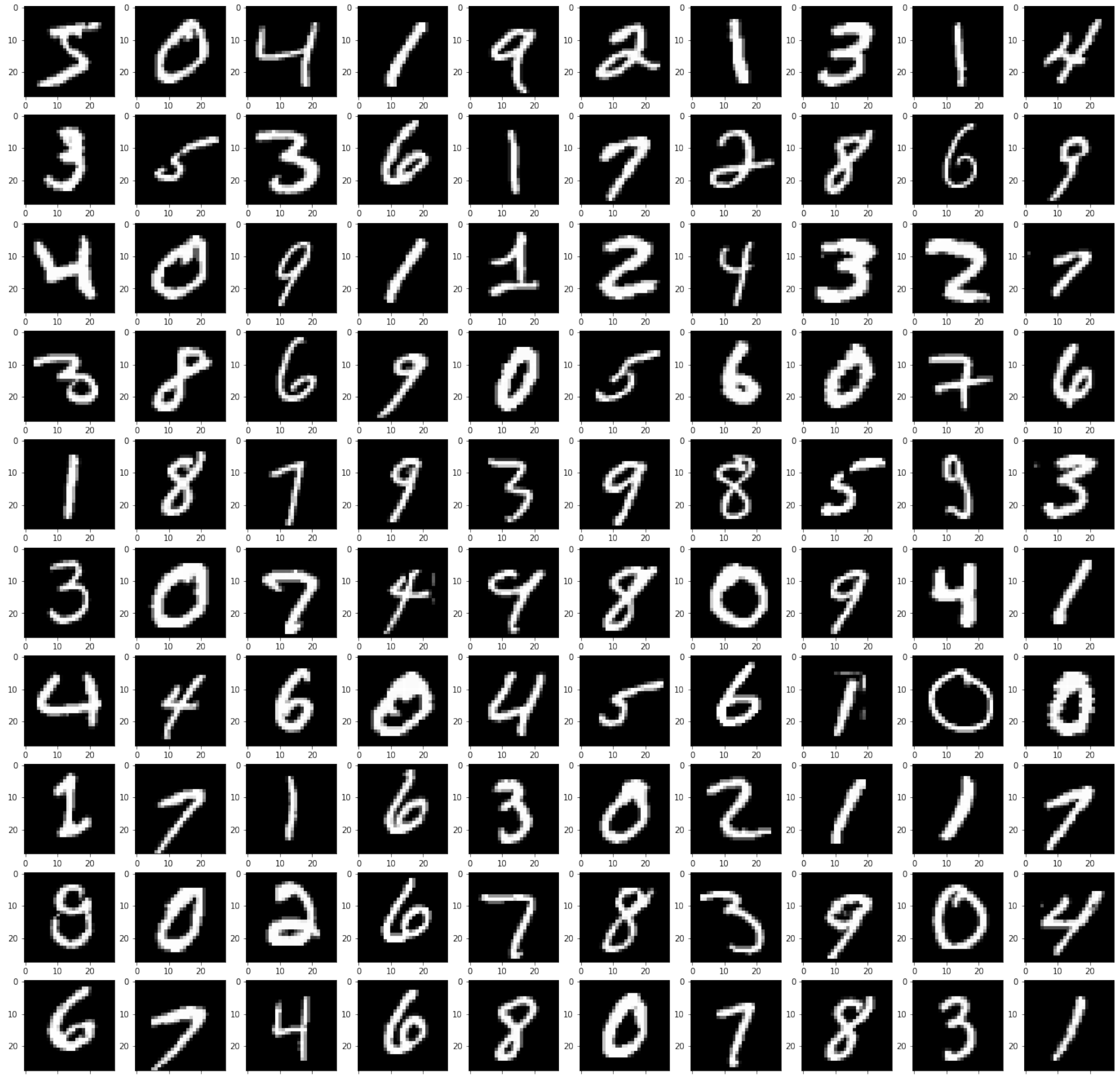


Figure F.1: Some samples of the original MNIST dataset, before conversion to 2class (or 4class)

G

fashion MNIST samples

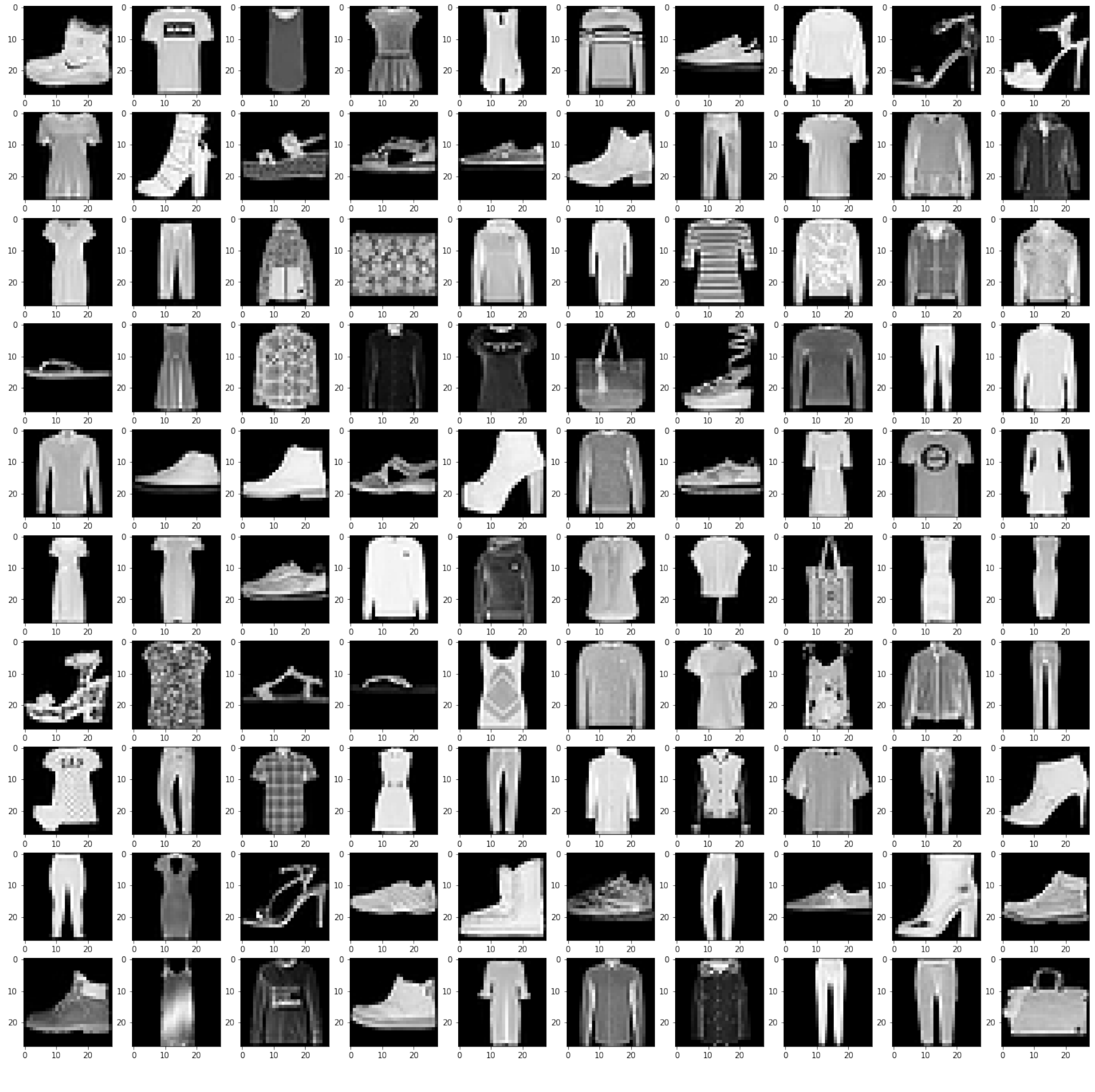


Figure G.1: Some samples of the fashion MNIST dataset