MSc thesis in Computer Engineering
5 Jan 2021

Aarhus University
Department of Computer Engineering

Federated Learning in Healthcare

Michael Quach - 201506330

Supervised by Christian Fischer Pedersen
Co-supervised by Christian Marius Lillelund

Preface and Acknowledgements

This thesis has been made for the degree of Master of Science in Computer Engineering at the Dept. of Eng. at Aarhus University. The source files can be found at: `https://github.com/micqu/ms4`.

First and foremost I would like to thank my supervisor Christian Fischer Pedersen and co-supervisor Christian Marius Lillelund for the continual guidance, knowledge and patience throughout the semester.

Furthermore, I would like to thank Aalborg Municipality and DigiRehab for providing the rehabilitation data set. I also very much appreciate the introduction to the domain of physiotherapy-based rehabilitation by Michael Harbo, DigiRehab.

Finally, I would like to thank my family and friends for their support during this work.

Michael Quach, Aarhus, Denmark
January 2021

Abstract

Introduction. In Denmark, elderly with decreasing functional capacity are offered rehabilitation programs to increase their self-reliance if they are deemed able to benefit from these programs. This is ultimately decided by caseworkers which may lead to differences in service quality across municipalities. This thesis is related to *AI Rehabilitation* (AIR), a signature project where a goal is to develop a solution in which caseworkers are supported in making this decision. However, when building a useful and powerful machine learning model, a lot of data is often gathered to a central location to allow model training. This can be problematic for privacy protection, which is important due to e.g. regulations such as the General Data Protection Regulation (GDPR) in EU. This thesis examines federated learning, a different machine learning approach that seeks to train a privacy-preserving model on decentralized data without requiring sensitive data to be shared. Three algorithms are evaluated: Federated Averaging, Federated Averaging with Differentially Private Stochastic Gradient Descent, and Federated Averaging with Secure Multi-Party Computation. **Methods**. To quantify predictive performance, comparative evaluations are carried out via objective metrics on four different data distributions across clients. Experimentation is first carried out on a benchmarking data set, MNIST, to prove functionality. These algorithms are subsequently evaluated on a data set regarding physiotherapy-based rehabilitation performed in municipalities of Denmark. **Results and Conclusion**. The results indicate that Federated Averaging generally performs well on four different data distributions for both data sets. The AUC scores on the rehabilitation data set were comparable or better than the conventional data-centralized model and locally trained models on the clients. Furthermore, the precision and recall curves for the federated models are generally comparable or better than the non-federated model. The differentially private algorithm performs with significantly lower accuracy compared to Federated Averaging on MNIST, but reaches the same AUC score on the rehabilitation data. The computation time of the differentially private algorithm is significantly higher than for the algorithm with SMPC (about a factor of 3 on the rehabilitation data). SMPC has no significant influence on model weights for high enough precision on the fixed precision encoding. Finally, the results show that the SHAP explanations are similar across the different algorithms. **Future work**. The experiments were conducted on a single compute node. It would be interesting to experiment with multiple nodes in a parallel setup to investigate the effect of communication delays and unreliable clients. For better performance on the rehabilitation data set, further hyperparameter optimization should be performed. Another interesting direction to investigate could be application of differential privacy in a non-federated data-centralized setting. Lastly, experiments were conducted of local differential privacy which provides the most private federated setting. It may be worthwhile to investigate global differential privacy in an attempt to attain better predictive performance while keeping or improving privacy guarantees.

Resumé

Introduktion. I Danmark tilbydes ældre med nedsat funktionsevne rehabiliteringsforløb for at øge selvhjulpenhed, hvis de anses for at kunne drage fordel af disse forløb. Dette besluttes i sidste ende af sagsbehandlere, som kan føre til forskelle i servicekvalitet på tværs af kommuner. Denne afhandling er relateret til *AI Rehabilitation* (AIR), et signaturprojekt, hvor et mål er at udvikle en løsning hvor sagsbehandlere støttes i at tage denne beslutning. Men når man bygger en nyttig og kraftfuld maskinlæringsmodel, indsamles ofte meget data til et centralt sted for at kunne træne modellen. Dette kan være problematisk for beskyttelse af privatlivets fred, hvilket er vigtigt bl.a. grundet databeskyttelsesforordningen (GDPR) i EU. Denne afhandling undersøger føderal maskinlæring, en anderledes metode, der forsøger at træne en privatlivsbevarende model på decentraliseret data uden at kræve deling af følsomme data. Tre algoritmer evalueres: Federeret Middelværdi, Federeret Middelværdi med Differentielt Privat Stokastisk Gradientafstamning, og Federeret Middelværdi med Sikker Flerpartsberegning. **Metoder**. For at måle prædiktiv præstation udføres sammenlignende evalueringer via objektive målinger på fire forskellige datafordelinger på tværs af klienter. Eksperimentering udføres først på et benchmarking datasæt, MNIST, for at eftervise korrekt funktionalitet. Disse algoritmer evalueres efterfølgende på et datasæt vedrørende fysioterapi-baseret rehabilitering udført i Danmarks kommuner. **Resultater og Konklusion**. Resultaterne indikerer at Federeret Middelværdi generelt fungerer godt på fire forskellige datadistributioner for begge datasæt. På rehabiliteringsdataene var AUC-værdierne tilsvarende eller bedre end værdierne for den datacentraliserede model og de lokalt trænede modeller på klienterne. Desuden er værdierne for precision og recall for de føderale modeller generelt tilsvarende eller bedre end den ikke-føderale model. Den differentielt private algoritme opnår signifikant lavere akkuratesse sammenlignet med den for Federeret Middelværdi på MNIST. Algoritmen når dog en tilsvarende AUC-værdi på rehabiliteringsdataene. Beregningstiden for den differentielt private algoritme er signifikant højere end for algoritmen med sikker flerpartsberegning (omkring 3 gange højere på rehabiliteringsdataene). Sikker flerpartsberegning har ingen signifikant indflydelse på modelvægtene ved brug af en tilstrækkelig stor præcision under præcisionskodningen. Endeligt viser resultaterne at SHAP-analyserne er tætte på tværs af de forskellige algoritmer. **Fremtidig Arbejde**. Eksperimenterne blev udført på en enkelt computer, og derved kunne det i stedet være interessant at eksperimentere på flere enheder i en parallel opsætning for at undersøge effekten af kommunikationsforsinkelser og upålidelige klienter. For at forbedre ydeevnen på rehabiliteringsdataene bør der udføres yderligere hyperparameteroptimering. Anvendelsen af differentielt privatliv på en datacentraliseret model kunne også være en interessant retning. Eksperimenter blev udført med lokalt differentielt privatliv, men det kan dog være umagen værd at undersøge globalt differentielt privatliv i et forsøg på at forbedre prædiktiv præstation og samtidig opretholde eller forbedre privatlivsgarantierne.

Contents

Chapter 1

Introduction

1.1 Background

In Denmark, elderly with reduced functional capacity are offered assistive devices as a remedy to improve their daily lives. Due to the reduced physical load that results from assistive devices, a citizen may experience that their functional capacity has decreased after a while. This in turn may lead to reliance on additional assistive devices, which over time can increase the need for municipal-provided home care.

In the beginning of 2015, a new law was introduced, namely §83a in the Danish Social Service Act (Serviceloven) [1]. The legislation requires that municipalities offer rehabilitation programs to citizens with reduced functional capacity, if the program is deemed beneficial for their functional capacity. The goal here is to allow citizens to become more independent, which is important due to challenges such as the increasing number of elderly, and the decreasing number of available social workers to take care of them [2].

Despite the physiotherapy-based rehabilitation offered by the municipalities, it has been the case that the rehabilitation has not benefitted citizens optimally on several occasions [3]. An analysis was made in 2019 by the benchmarking unit of the *Ministry of Social Affairs and the Interior* which showed that there is an imbalance in the number of rehabilitation offers among 17 municipalities [4]. The authors write that this could be due to differences in demography, or due to socio-economic reasons, or service quality in general. And finally, they mention the differences could stem from variation in visitation practices (applicability assessment of a rehabilitation program for individual citizens). For instance, the parameters that are used to measure the citizens' potentials may differ across municipalities, along with the "threshold" used to decide when a citizen will benefit from a rehabilitation program [4], which may be influenced by subjectivity.

In an attempt to raise the quality for the future Danish public sector, the Government, Local Government Denmark (KL), and Danish Regions has agreed to launch 15 signature projects to gain experience with artificial intelligence in the period of 2019 to 2022 [5]. This thesis is related to one of these projects, *artificial intelligence in rehabilitation* (AIR), that tries to develop a solution in which caseworkers are supported in their assessment of the citizens' needs with respect to physical training, use of assistive devices, and fall prevention. It is

assumed that offering physiotherapy-based rehabilitation for citizens can reduce the need for home care provided by the municipalities, thus increasing self-reliance among elderly citizens - adding to life quality, and furthermore, a more sustainable economy for the society [3].

1.2 Motivation

When building a useful and powerful machine learning model it is often the case that a lot of data is needed to train it [6]. Moreover, conventional machine learning methods require that data is gathered to a central location in order to learn from it [7] - adding risk in case a data leakage occurs, since it may leak all the gathered data. Another problem with data collection is the possible violation of user privacy when the data is sensitive, which is often the case with healthcare data.

In EU, the General Data Protection Regulation (GDPR), implemented in May 2018, protects the data subjects, or users, by regulating e.g. how personal data is processed (Art. 1 (1) in [8]). For instance, processing is lawful when the data subject has given consent to processing, or when processing is necessary to perform a contract with the data subject (Art. 6 (1) in [8]). To discourage infringement of GDPR, large fines can be imposed depending on the circumstances of each individual case (Art. 83 in [8]).

In light of the above, it may be worthwhile to investigate other types of machine learning techniques in the domain of physiotherapy-based rehabilitation. Particularly, in this thesis, there will be a focus on a relatively new setting in the machine learning field - Federated Learning.

Federated Learning is a machine learning approach that seeks to build privacy-preserving models on decentralized data without requiring sensitive data to be shared. The model training happens mainly on the client-side containing the sensitive data, and instead of sending the sensitive data directly to a service provider, only model updates are sent to be aggregated [9]. The method has been applied in production by, for instance, Google for their GBoard (Google keyboard) which makes query suggestions on smart devices [9, 10], and also Apple for their QuickType keyboard and personalized "Hey Siri" vocal classification [9, 11]. Federated Learning was initially introduced for mobile devices, but soon expanded into cross-silo settings where data is siloed among multiple organisations [9]. This is the case in the domain of physiotherapy-based rehabilitation, where multiple municipalities want to use algorithms on their own sensitive data while avoiding implications of privacy.

Although federated learning significantly improves privacy versus a data-centralized approach, federated learning is not perfectly privacy-preserving by itself [12–15]. Currently, the most prevalent methods for providing privacy guarantees for federated learning is through differential privacy and cryptographic methods such as Secure Multi-Party Computation (SMPC), and homomorphic encryption [9,16,17]. Later, differential privacy and SMPC will be described and applied in a federated setting.

1.3 Problem Formulation

The focus of the present master's thesis is to design, implement, test, compare, and document the already existing Federated Averaging algorithm and variants thereof for training neural networks on decentralized data with applications to clinical decision support in physiotherapy-based rehabilitation. Additional privacy-enhancing techniques such as differential privacy and secure multi-party computation should be considered. The algorithms must be optimized to (1) learn and predict on a benchmarking data set to prove functionality, and (2) learn and identify citizens who are most likely to complete a physiotherapy-based rehabilitation program successfully. Thus, concrete experiments must be conducted. To quantify predictive performance of the algorithms, comparative evaluations should be carried out via objective metrics under various federated learning contexts vs. a non-federated and data-centralized context. In the comparisons, computation time and explainability should be considered. As an overall assessment of the algorithms and the decision support system, comparisons with state of the art should be conducted if similar algorithms have been applied.

Chapter 2

State of the Art

2.1 Machine Learning in Healthcare

As mentioned earlier, machine learning can help care givers make better decisions for the benefit of the patients, and also improve the economy of the society. To be of relevance and to allow such benefits, the decision support system must have good inference performance to perform well, meaning that the system must make good predictions.

Tack [18] reviewed applications of supervised and unsupervised machine learning for musculoskeletal medicine in 2019. The author concludes that the super-human level diagnostics, decision making and measurements of supervised machine learning has potential to enhance physiotherapy practice, and physiotherapists should apply technological innovation in practice.

An example of a study that resulted in good inference performance is by Nijeweme-d'Hollosy et al. [19] in 2018. They explored the possibilities of using supervised machine learning for a clinical decision support system. The purpose of the system was to support patients with low back pain, in their self-referral to primary care. The authors defined a three-class classification problem, and compared performance between three tree-based classification models: decision trees, random forests, and boosted trees. Training was done using a 70%/30%-split of training and validation data based on fictive cases of low back pain. Additionally, data of real-life cases were collected for the test data set. Model performances were evaluated using confusion matrices, and metrics such as accuracy, sensitivity, specificity, precision, and kappa (a metric that can compare the observed accuracy with the expected accuracy [20]). The total observed accuracies were 71%, 53%, and 71% respectively. The authors concluded that using machine learning for their decision support system showed promising results albeit more work was needed.

Another example is Shahmoradi et al. [21] in 2019 where a decision support system was developed to help clinicians predict the efficacy of neurofeedback therapy for patients with Attention Deficit Hyperactivity Disorder (ADHD) in Tehran. This somewhat resembles the setting in this thesis of physiotherapy-based rehabilitation in Denmark. The authors of the article employed supervised training of an artificial neural network with two hidden layers to perform binary classification. The number of hidden layers was fixed and then the number of neurons was optimized using a 70%/10%-split of training and validation data, where the

remaining 20% of the data was held out for testing. Model performance was measured using a Receiver Operating Characteristic (ROC) curve, confusion matrices, and metrics such as sensitivity, specificity, and the area under the ROC curve (AUC). Principle Component Analysis (PCA) was used to reduce data dimensionality. A technique named SMOTE was also employed to balance their class-imbalanced data set. Their study resulted in a model with 100% sensitivity on the test set - indicating good performance.

2.2 Federated Learning in General

Federated Learning resembles distributed machine learning where an optimization problem is solved by many compute nodes. When the amount of data is very large, a single compute node would not suffice to even store the data. However, distributed machine learning focuses more on increasing performance in a compute cluster where computation between processor and memory is low. On the other hand, federated learning focuses more on learning across different nodes in a network where communication costs are much greater, and the connectivity may be more unreliable. Moreover, in the federated case, data is often dissimilar across devices which stands in contrast to the distributed case. [22] Another consideration is that the privacy is often the focal point in a federated learning setting.

A survey on federated learning systems by Li et al. [16] showed that most federated learning studies focus on state-of-the-art models such as neural networks, and federated stochastic gradient descent. Other types of widely used models include tree-based methods (e.g. Federated Forest [23] based on the traditional random forest), and linear models such as linear regression, logistic regression, and support vector machines.

Some of the federated algorithms that focus on effectiveness are based on Stochastic Gradient Descent (SGD), while others are designed for specific models such as neural networks, trees, and others [16]. SGD can be implemented in a federated setting by simply calculating a single gradient update each training round. However, this may lead to a large number of training rounds if convergence is slow. A simple and popular algorithm for federated training is Federated Averaging that is based on SGD. Federated Averaging reduces the amount of training rounds by letting clients compute multiple gradient updates each round [16, 24]. Another algorithm, designed specifically for neural networks, is Probabilistic Federated Neural Matching [16, 25] that exploits the model structure to combine local client models in a clever way. The Federated Averaging algorithm assumes that each client model is initialized from the same random initialization, and the model updates are aggregated in the order of the neural network weights. The Probabilistic Federated Neural Matching algorithm constructs a global model by grouping and combining neurons in collections of feedforward neural networks. Thus, this method is also applicable for combining pre-trained models as well.

The centralized design is often studied in the literature [16], where a server is needed to orchestrate the federated learning process. The central orchestrator is potentially a single point of failure [9], and if a trusted server can not be determined, then a fully decentralized (peer-to-peer) approach may be of interest.

A question that may arise when working with federated learning is whether or not a global model for all clients (consensus solution [17]) is actually a good solution. One of the challenges

in federated learning is that the data found across clients is not independent and identically distributed (non-IID) [9]. For instance, assume that the type of citizens, who are predicted to complete a rehabilitation program, is correlated with municipality. This would mean that a model trained in one municipality may not perform well if used in a different municipality, because they are trained on data from different types of citizens. If the data is highly non-IID, it is better to create a personalized model for each client (pluralistic solution [17]) [9]. Creating personalized models could be done using a technique such as Model-Agnostic Meta-Learning where new local patterns on each client can be learned quickly [26].

2.3 Federated Learning in Healthcare

The possibility of privacy-preserving machine learning models that can utilize siloed data has sparked interest in federated learning for clinical settings.

Pfohl et al. [12] studies the efficacy of a data-centralized learning approach versus a federated learning approach versus local learning on each client, in both private and non-private settings. In the private setting they used differential privacy to attain privacy guarantees. They considered logistic regression using feedforward neural networks to perform binary classification of prolonged length of stay and in-hospital mortality. The authors found that federated learning performs comparably to data-centralized learning in their case. Moreover, they found that differentially private centralized training attains good privacy guarantees with a slight reduction in AUC. Furthermore, they show that models trained using differentially private federated learning performed poorly in terms of AUC and ϵ (privacy budget).

Similarily, Sharma et al. [27] studies in-hospital mortality using a federated averaging approach. Logistic regression and a feedforward neural network classifier were employed in both a data-centralized and federated setting. They measured model performance using AUC and the area under the precision-recall curve, and find that the federated models perform comparably to models trained in a data-centralized setting.

2.4 Privacy in Federated Learning

In terms of privacy, federated learning can, as previously mentioned, increase privacy by only sending model updates to the server. The model updates should contain as little information as possible to solve the specific training task, and they should be stored ephemerally and aggregated as soon as possible [9]. This approach can help reduce the risks of a large data leakage, however, it has been shown that training data extraction is still possible due to memorization (overfitting) [13]. Another problem is model inversion attacks, that aims to reconstruct data using model parameters [14]. Moreover, a black-box approach, without model parameters, can also reveal information to an adversary through membership attacks [15], that can determine the presence of a particular data record in the training data set. An apparent solution may be to anonymize the data set before model training but this is not guaranteed to provide privacy against e.g. de-anonymization attacks [28]. Currently, the most prevalent methods for providing privacy guarantees is through differential privacy and cryptographic

methods such as Secure Multi-Party Computation (SMPC), and homomorphic encryption [9, 16, 17].

2.5 Explainability in Federated Learning

In a real-life clinical scenario, the users of a decision support system must be able to trust the model and its predictions in order to use the predictions in practice. A way of increasing user trust is to explain to the user what the model bases its predictions on. In federated learning each client usually have a model available locally, meaning that there is usually no need to perform communication over a network during inference time. Additionally, one can use methods such as e.g. SHAP or LIME to explain predictions locally [29].

2.6 Other Non-Centralized Methods

Private Aggregation of Teacher Ensembles (PATE) is a privacy-preserving machine learning framework, that consists of mainly three parts: an ensemble of teachers, an aggregation mechanism, and a student model [30]. The ensemble of teachers are models that are each trained on a disjoint part of the private data set. Any type of model can be used in PATE, as the framework is based on using only the predictions of the teacher models. The aggregation mechanism takes the outputs of the teachers and creates a histogram of the label votes. Afterwards, noise is added to this histogram and then the highest voted label is picked. The type of noise selected in the earlier version of PATE was Laplace noise, however, in an improved version, the authors use Gaussian noise as the tails of the distribution drop faster. To analyse their new mechanisms using Gaussian noise, they employ Rényi differential privacy instead of the commonly used (ϵ, δ)-differential privacy as analysis framework. The student model tries to learn the labels of an unlabeled and public data set by querying the teachers.

An interesting part of PATE is the analysis of the privacy bound ϵ. When the teachers disagree on an answer such that there is no real consensus, then the privacy cost is higher. If that is the case, or the student is confident in its own answer, then the aggregated answer is omitted by PATE. The more the teacher predictions agree, the more privacy is preserved.

It is here assumed that the unlabeled data set from the student is publicly available, meaning that it is not a privacy issue to send this data to the teachers. It may be possible to use cryptographic methods such as homomorphic encryption and SMPC to secure the data.

Another type of privacy-preserving machine learning is Split Learning introduced in 2018 [31]. This technique can allow training of deep neural networks across multiple data sources while keeping the sensitive training data private. The idea of Split Learning is to split a deep neural network at a certain layer such that two parts of the neural network is created. Then, one part is used by the client and the other part is used by the server. To train the network the client must make a forward pass using its local sensitive data. The output at the split is then sent to the server for further processing where the backpropagation algorithm can be applied to update the model weights. When the backpropagation reaches the split, the computation is again carried out at the client side. This process can also be applied over multiple clients. A

benefit of such an approach is that information that needs to be sent is reduced, because only values at the split needs to be sent.

2.7 Summary

In this case of physiotherapy-based rehabilitation focus will be laid on a non-peer-to-peer federated learning approach using neural networks. Inspired by the state of the art, the algorithms that will be experimented with are the following: Federated Averaging, Federated Averaging with Differential Privacy, and Federated Averaging with Secure Multi-Party Computation. Model performance will be compared in terms of computation time, accuracy metrics such as precision, recall and AUC. As it is often the case that data distribution changes across clients, it is relevant to compare the algorithms on different data distributions as well. Finally, to both compare algorithms and show feasibility of explainability, SHAP will be used.

Chapter 3

Machine Learning Theory

In this chapter, some preliminary supervised machine learning theory will be described before later introducing federated learning, differential privacy, and secure multi-party computation. In the end of this chapter several methods will be described for evaluating different machine learning algorithms.

3.1 Neural Networks

As neural networks are often considered state-of-the-art [16,32] for different tasks, e.g. language models [33] and healthcare [12,27], we here also apply neural networks. Since neural networks will be used during experimentation, it is relevant to review how neural networks work and how they learn.

3.1.1 Neuron

A neural network is built using neurons. To describe what a neuron is in terms of neural networks, it may be beneficial to first describe from where they are inspired to get an intuition of how they function. In biology neurons are a fundamental part of the brain. This type of cell is capable of receiving, sending, transforming, and relaying electrical signals. The three main parts of a neuron are its *dendrites*, *axon*, and cell body or *soma*.

The dendrites branch outwards to other cells and are able to receive signals from them. When this happens, the signals travel across the soma and the long thin axon connected to it. The axon ends in multiple terminal buttons that can send signals to dendrites of other neurons. This junction between two neurons is called the *synapse*, and the gap between terminal buttons and dendrite neurons is called the *synaptic cleft* (or gap). When a neuron receives signals from other neurons, the voltage level of the neuron (membrane potential) changes which may cause it to *fire* and send a signal to other neurons. In order to do this, the membrane potential must reach a certain threshold [34–36]. An illustration of a biological neuron can be seen in Figure 3.1.

Similarly to a biological brain, an artificial neural network also consists of neurons. These

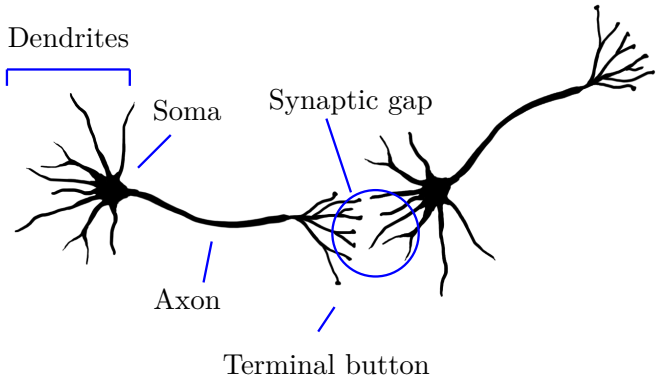


Figure 3.1: Illustration of a biological neuron.

neurons take as input a number of values $x_d, d = 1..D$, forming a vector $\mathbf{x} \in \mathbb{R}^D$ where D is the number of input values - resembling the dendrites of a biological cell. Then, multiplying $\mathbf{x}$ with a weight vector $\mathbf{w}$ followed by an addition of a bias term gives a scalar value p. The result p (*preactivation*), is then passed through a so-called *activation function*, f, that determines whether the neuron fires. This results in a prediction $\hat{y}$ of the true label y (activation functions will be explained in the coming section). The final expression is shown in (3.1) [37].

$$\hat{y} = f(p) = f(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}), \tag{3.1}$$

where

$$\tilde{\mathbf{w}} \leftarrow \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \ \tilde{\mathbf{x}} \leftarrow \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}. \tag{3.2}$$

Following the same notation as [37], the augmented versions of the weight and sample vectors have been used to allow for more compact notation. In the following sections $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{x}}$ will be referred to simply as $\mathbf{w}$ and $\mathbf{x}$ respectively. An illustration of an artificial neuron can be seen in Figure 3.2.

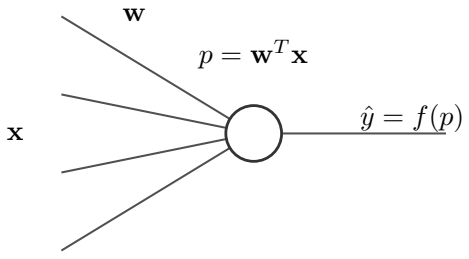


Figure 3.2: An artificial neuron that mimics a biological neuron.

A neural network can then be formed by stacking multiple layers of neurons. Figure 3.3 shows a three layer neural network where the first and last layers are called the input and output layers respectively. The layers between these are called hidden layers [37]. In this case there is only one hidden layer, but an arbitrary amount of layers with different numbers of neurons can

be inserted. Data enters through the input layer and gets transformed while passing through the network until it reaches the output layer. If there are no feedback connections such that output is fed to input, then the network is called a *feedforward* neural network or *Multi-Layer Perceptron* (MLP) [38] because information only moves in one direction from the input layer to the output layer. An example of a three-layer neural network can be seen in Figure 3.3.

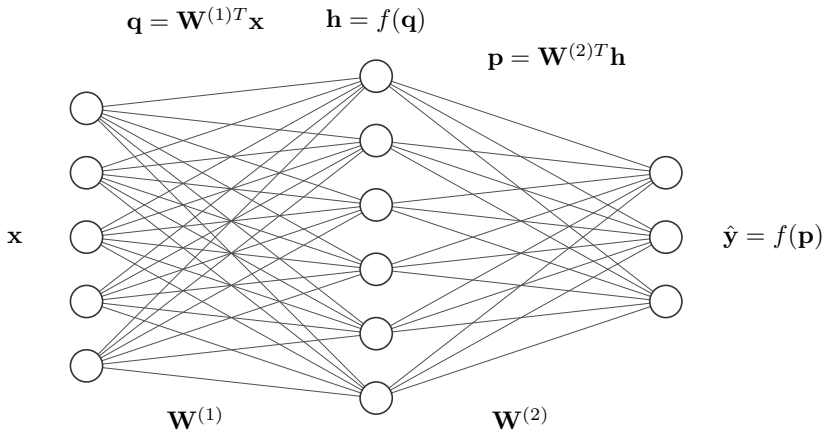


Figure 3.3: A three-layer neural network.

Here we see that the hidden layer (middle layer) can be calculated as:

$$h_k = f(\mathbf{W}_k^{(1)T}\mathbf{x}),\ k = 1..K, \tag{3.3}$$

where K is the number of neurons of $\mathbf{W}^{(1)}$, k indicates the particular neuron of the hidden layer, and $\mathbf{W}_k^{(1)T}$ is the k^{th} row vector of $\mathbf{W}^{(1)T}$. This can be rewritten into a more concise matrix form:

$$\mathbf{h} = f(\mathbf{W}^{(1)T}\mathbf{x}), \tag{3.4}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{M \times N}$, and M and N indicates the number of neurons in the input layer and hidden layer respectively. The output, or prediction, of the three-layer network can then be calculated as:

$$\hat{\mathbf{y}} = f(\mathbf{p}) = f(\mathbf{w}^{(2)T}\mathbf{h}) = f(\mathbf{w}^{(2)T}f(\mathbf{W}^{(1)T}\mathbf{x})). \tag{3.5}$$

3.1.2 Activation Function

Activation functions can introduce non-linearity in a neural network. Without activation functions, the neural network turns into a chain of linear functions which altogether will be a linear function as well. If the problem to which we apply the model is linear, then a linear network is sufficient. For instance, determining travelled distance given travel time can be seen as linear. The classification problems that are experimented with later, i.e. image classification and rehabilitation completion, are not linear problems. For non-linear problems, the activation functions must be different from the identity function $f(x) = x$.

There exist many different activation functions but common for all of them is that they must be differentiable, as we will see in the coming section about learning using backpropaga-

tion. Some commonly used activation functions have been listed in Table 3.1 and plotted in Figure 3.4.

Name	Function	Derivative
Identity	$f(x) = x$	$f'(x) = 1$
Logistic	$f(x) = \frac{1}{1+\exp(-x)}$	$f'(x) = f(x)(1 - f(x))$
ELU	$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$	$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ \alpha \exp(x), & \text{if } x < 0 \end{cases}$
ReLU	$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$
Softmax	$f_i(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{k=1}^{K} \exp(x_k)}, i = 1..K$	$\frac{\partial f_i(\mathbf{x})}{\partial x_j} = f_i(\mathbf{x})(\delta_{ij} - f_j(\mathbf{x}))$

Table 3.1: Commonly used activation functions.

A constant $\alpha > 0$ should be used for ELU, where a larger α value results in greater negative values when $x \leq 0$. The Softmax activation function is a generalization of logistic regression that can be used for classification problems with multiple classes [39]. It is often used in the final layer of a neural network as the outputs resemble probabilities, thus improving interpretability. It can be seen that the numerator is an exponential function that is positive for any real number x_i. The denominator is a sum of similar exponential functions that includes the nominator, meaning the values of vector $\mathbf{x}$ is scaled to the range $[0, 1]$ such that they sum up to 1. As an example, Softmax could output $\hat{\mathbf{y}} = \Big[0.1, 0.8, 0.1\Big]$. Softmax is a vector function that takes a vector as input, and outputs a vector of similar size. Therefore, the derivative uses the partial derivates of each component with respect to each component. Here, δ_{ij} is the Kronecker delta function [40] as shown in (3.6).

$$\delta_{ij} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases}. \tag{3.6}$$

Although Softmax has not been plotted in Figure 3.4 it is similar to the logistic activation function when the number of classes $K = 2$ [39].

3.1.3 Loss Function

Previously, the process of passing data through a neural network, a *forward pass*, was described. To evaluate the prediction performance of the forward pass, the output $\hat{\mathbf{y}}$ is passed through a loss function that calculates how far the prediction is from the given target $\mathbf{y}$. We would like to minimize the loss function such that the prediction gets closer to the target – allowing the model to learn.

Some popular loss functions for classification tasks include *Binary Cross-Entropy* and *Categorical Cross-Entropy*, which will be used during experimentation. Here, $\mathbf{y}$ indicates the true

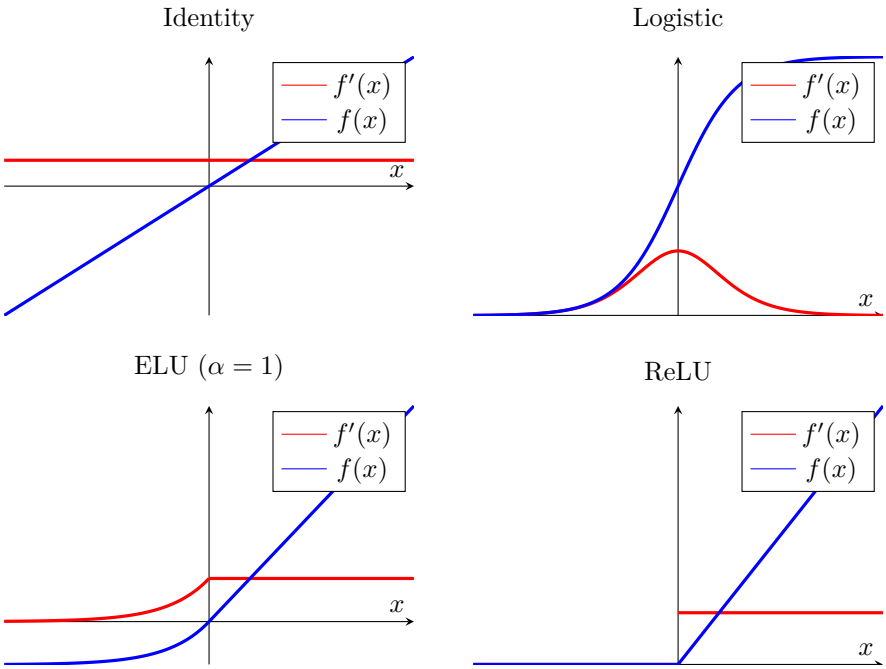


Figure 3.4: Plots of several activation functions and their derivatives.

label using one-hot encoding - a vector of only ones and zeros where ones indicate indexes corresponding to the correct classes. For instance, a vector $\mathbf{y} = \begin{bmatrix}0, 1, 0\end{bmatrix}$ indicates that the second class is the correct class. Again, $\hat{\mathbf{y}}$ is the probability-estimates - e.g. Softmax output.

Cross-Entropy

When the total number of classes $K = 2$, the binary cross-entropy loss can be calculated as:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K=2} y_k \log(\hat{y}_k) = -y_1 \log(\hat{y}_1) - (1 - y_1) \log(1 - \hat{y}_1). \tag{3.7}$$

This loss is illustrated in Figure 3.5 where the loss decreases when the prediction is near the true value. But, when the prediction is very confident in a wrong prediction (in this case when $\hat{y}$ goes towards zero) the loss increases rapidly.

In the cases where the number of classes $K > 2$ it is common to use categorical cross-entropy instead [41]:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k). \tag{3.8}$$

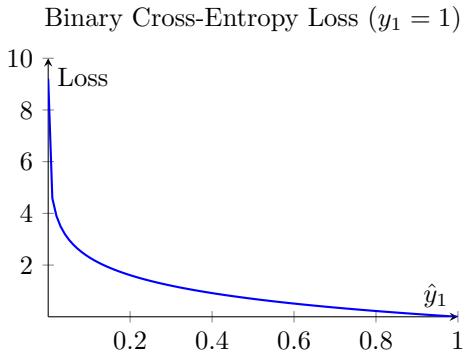


Figure 3.5: Binary cross-entropy loss when $y_1 = 1$.

3.1.4 Gradient Descent

Gradient descent is an algorithm often used to minimize loss functions such as cross-entropy mentioned previously. It works in an iterative fashion to find local extrema. If we consider an objective function f and a starting point $\mathbf{w}$, corresponding to the weights of a neural network, then the gradient at $\mathbf{w}$ is:

$$\nabla f(\mathbf{w}) = \begin{bmatrix} \frac{\partial f}{\partial w_1}(\mathbf{w}) \\ \vdots \\ \frac{\partial f}{\partial w_n}(\mathbf{w}) \end{bmatrix}. \tag{3.9}$$

This gradient points in the direction of the largest increase of f at $\mathbf{w}$. In addition, the negative gradient, $-f(\mathbf{w})$, points in the direction of the largest decrease [42]. This leads to the following algorithm:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}), \tag{3.10}$$

where η is a positive scalar called the *step size* or *learning rate*. The idea is to update the value of $\mathbf{w}$ by following the direction of the negative gradient until some stopping condition is met. For instance, when $\mathbf{w}$ changes very slowly, or until the gradient gets closer to zero as $\mathbf{w}$ approaches the local extremum. Figure 3.6 depicts several gradient descent steps on a two-dimensional optimization problem.

Usually a small value, e.g. 0.03, is used for η because a high value may jump past the local optimum and never settle completely. Though, selecting an η value that is too small results in excessive gradient evaluations and slower progression.

3.1.5 Learning using Backpropagation

A neural network is usually initialized with random weights which is unlikely to perform well at any real task. A way of improving performance is by carefully adjusting the weights using gradient descent such that the loss function is minimized. I.e. pass a sample vector $\mathbf{x}$ through the network (forward pass) and get a prediction $\hat{\mathbf{y}}$. Then, compare $\hat{\mathbf{y}}$ with a target vector $\mathbf{y}$ by

[1]Plotting code found at: `https://tex.stackexchange.com/questions/544796`.

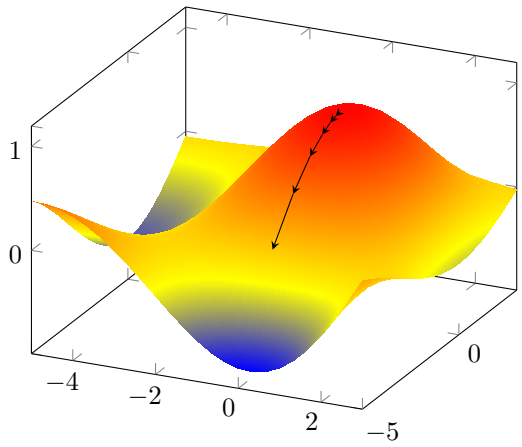


Figure 3.6: Example of gradient descent on a two-dimensional optimization problem. Arrows show gradient descent steps.[1]

calculating the loss/error. Finally, update the weights depending on the error such that the introduction of the same sample vector $\mathbf{x}$ to the model will yield a smaller error. [37]

As an example we can use the following optimization criterion (loss function), that calculates the loss given all the trainable parameters in the network $\mathbf{w}$:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2 = \frac{1}{2}\left\|\mathbf{y} - \hat{\mathbf{y}}\right\|_2^2. \tag{3.11}$$

The criterion is optimized using gradient descent, meaning that the derivative of $\mathcal{J}$ needs to be calculated with respect to all parameters $\mathbf{w}$. The backpropagation algorithm allows this derivative to be calculated efficiently by using the chain rule for composite functions. As an example, let f and g be two functions that map from a real number to a real number, and let x be a real number. I.e. $f, g : \mathbb{R} \to \mathbb{R}$ and $x \in \mathbb{R}$. Then suppose that $z = f(y) = f(g(x))$ where $y = g(x)$. By the chain rule, dz/dx can be calculated as [38]:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{3.12}$$

The backpropagation algorithm uses the chain rule at the output layer of the network and applies it repeatedly towards the input layer. Using the three-layer neural network shown on Figure 3.3, the derivative of $\mathcal{J}$ with respect to the weights $\mathbf{W}^{(2)}$ is:

$$\frac{\partial \mathcal{J}}{\partial W_{jk}^{(2)}} = \frac{\partial \mathcal{J}}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial p_k}\frac{\partial p_k}{\partial W_{jk}^{(2)}}. \tag{3.13}$$

The first factor on the right-hand side is the derivative of the loss function $\mathcal{J}$ with respect to the output of the neural network $\hat{\mathbf{y}}$. By substituting the loss function $\mathcal{J}$ we get:

$$\frac{\partial \mathcal{J}}{\partial \hat{y}_k} = \frac{\partial}{\partial \hat{y}_k}\left(\frac{1}{2}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2\right) = \hat{y}_k - y_k. \tag{3.14}$$

The second factor is the derivative of the output of the neural network $\hat{\mathbf{y}}$ with respect to the preactivation value $\mathbf{p}$. Since $f(\mathbf{p}) = \hat{\mathbf{y}}$ we get:

$$\frac{\partial \hat{y}_k}{\partial p_k} = \frac{\partial f}{\partial p_k}(p_k) = f'(p_k). \tag{3.15}$$

We see here that the activation function f must be differentiable. For the last factor, the derivative of the preactivation $\mathbf{p}$ is calculated with respect to the weights $\mathbf{W}^{(2)}$ as:

$$\frac{\partial p_k}{\partial W_{jk}^{(2)}} = \frac{\partial}{\partial W_{jk}^{(2)}}(W_{jk}^{(2)T} h_j) = h_j. \tag{3.16}$$

These three factors can then be substituted back into (3.13) to get:

$$\frac{\partial \mathcal{J}}{\partial W_{jk}^{(2)}} = (\hat{y}_k - y_k) f'(p_k) h_j. \tag{3.17}$$

Finally, the weights $\mathbf{W}^{(2)}$ can be updated using:

$$W_{jk}^{(2)} \leftarrow W_{jk}^{(2)} - \eta \frac{\partial \mathcal{J}}{\partial W_{jk}^{(2)}}. \tag{3.18}$$

The process described above must then be followed to update the weights in $\mathbf{W}^{(1)}$. It will not be shown here, but doing this we will find that the first two factors calculated above can be cached for future calculations. This is because the weights in $W^{(1)}$ affects the hidden layer neurons, which in turn affects the loss function through all values of $W^{(2)}$ [37]. Calculating backwards reduces computation because the intermediate terms are used in derivatives for previous layers.

Variants of Gradient Descent

Optimization algorithms that use a single sample at a time to calculate the gradient and update the weights, are sometimes called *stochastic*. Algorithms that use all samples are called *batch* because the training samples are all processed at the same time in a large batch. Often, algorithms use the so-called *mini-batch* method which uses more than a single sample but fewer than all samples. [38]

That is, instead of using e.g. Stochastic Gradient Descent (SGD) where $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{x}, \mathbf{w})$, the following is done instead [43]:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \frac{\mathbf{1}}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} f(\mathbf{x}^{(i)}, \mathbf{w}). \tag{3.19}$$

The gradient is calculated as usual with respect to $\mathbf{w}$, however, it is now calculated over a batch of samples $\mathcal{B}$. Larger batch sizes $|\mathcal{B}|$ gives more accurate estimates of the gradient but

with diminishing returns. Furthermore, the available memory may limit the batch size if the samples are to be processed in parallel. Using smaller batches can provide a regularizing effect but this may require a smaller learning rate to counter the higher variance of the gradient estimate [38]. Mini-batch SGD will be used during experimentation, and different batch sizes will be applied as this can have a great impact on e.g. accuracy metrics.

3.1.6 Labeling

For a multiclass classifier, the probability estimates, $\hat{\mathbf{y}}$, from a neural network can be turned into labels using the arg max function as seen in (3.20). Here arg max finds the largest component of vector $\hat{\mathbf{y}}$ and returns the index i of that component. This makes sense, as the largest component of $\hat{\mathbf{y}}$ indicates the class of which the model is most confident.

$$l = \arg\max_i \ \hat{y}_i. \tag{3.20}$$

As an example, let $\hat{\mathbf{y}} = [0.7, 0.1, 0.2]$ where the sum of the elements equals 1, i.e. $\sum_i \hat{y}_i = 1$, then the resulting label would be 1 to indicate the first element. For a binary classification problem, the model output is usually just a single probability. To turn this into a binary classification, a threshold can be specified. For instance, using the labels 0 and 1, and a threshold $th = 0.5$, the predictions less than th can be assigned to class 0 and 1 otherwise. I.e. we have:

$$l = \begin{cases} 0, & \text{if } \hat{y} < th \\ 1, & \text{if } \hat{y} \geq th \end{cases}, \ th = 0.5. \tag{3.21}$$

3.1.7 Generalization Performance

The goal of training a machine learning model is not high prediction performance on the training data. Rather, we are interested in maximizing some performance measure that is based on unseen data. As it may be difficult to optimize for this measure directly, it is done indirectly by optimizing the training error. It is commonly the case that the available data is split appropriately into two non-overlapping parts - a larger training set, and a smaller *validation* set. The training set will then be used to update the model parameters, and the validation set will be used to measure how well the model generalizes to unseen data. It is assumed here that the validation set is similar to new data, i.e. the set should be a representative sample of the true population [37, 38]. During later experimentation the data will be shuffled randomly before splitting the data, such that the training data is likely to be similar to the validation data.

3.2 Regularization

To prevent a neural network model from overfitting to the training data, and increase generalization performance, one can apply regularization methods such as e.g. ℓ_1, ℓ_2, or *dropout*. During experimentation, dropout [44] is used as a regularizer.

Dropout works by disabling, or dropping out, neurons and connections temporarily during the training phase of the network. The probability of a dropout is set to p, where $p = 0.5$ is close to optimal for many networks [44]. After training, all neurons and connections are enabled again, however with the weights scaled by p. Describing this for the feedforward process we have:

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \end{aligned} \tag{3.22}$$

where $l \in \{0, ..., L-1\}$ indicates the index of the layers in a neural network, where $\mathbf{y}^{(l)}$ is the output of layer l and $\mathbf{y}^{(0)} = \mathbf{x}$ corresponds to the input. Here $*$ denotes element-wise multiplication. $\mathbf{r}^{(l)}$ is a vector where each element $r_j^{(l)}$ is Bernoulli distributed with probability p. This means that each element is 1 with a probability of p, and 0 with a probability of $q = 1 - p$. Finally, $\tilde{\mathbf{y}}^{(l)}$ is the *thinned* output of layer l. After applying this process on every layer during the forward pass, the backpropagation algorithm can be applied on this thinned network. The main idea of using dropout, is to regularize by averaging the predictions of many models. For large neural networks it may be prohibitive to average predictions from many of these models. Dropout serves as an approximation to efficiently combine exponentially many different neural network architectures; 2^n where n is the number of neurons, since every neuron is either enabled or disabled. [44]

3.3 Evaluating Algorithms

In this section several methods are described that are later used to evaluate performance of different algorithms.

3.3.1 Confusion Matrix

A confusion matrix is a table that describes how a classification model performs. Such a table is made by listing the classification predictions and actual (true) classifications respectively along the axes of a matrix. The predictions are then grouped into these cells and counted. Thus, a good classifier will have higher values along the diagonal and lower values in off-diagonal entries. A perfect classifier has zeros in all off-diagonal entries.

The structure of a confusion matrix for a binary classifier is shown in Figure 3.7. True positives and true negatives correspond to correct predictions since the predicted values are the same as the actual values. A false positive corresponds to incorrect positive predictions (Type I error), and false negatives correspond to incorrect negative predictions (Type II error).

		Actual: True	Actual: False
Predicted	True	True Positives (TP)	False Positives (FP)
	False	False Negatives (FN)	True Negatives (TN)

Figure 3.7: Confusion matrix for a binary classifier.

3.3.2 Accuracy

A simple way of measuring model performance is by calculating the accuracy as follows:

$$\begin{aligned} \text{Accuracy} &= \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \\ &= \frac{TP + TN}{TP + TN + FP + FN}. \end{aligned} \tag{3.23}$$

If there is a significant imbalance in the number of true and false labels, then accuracy may not be a useful metric. For instance, if a given model tries to predict an unlikely event, such as a large earthquake, the model could simply always predict "False". It would give a high accuracy score but this would not be very useful.

3.3.3 Precision, Recall and Specificity

For class-imbalanced data it may be more beneficial to use metrics such as *precision*, *recall*, and *specificity* [45]. Precision is calculated as the proportion of positive predictions that are actually correct.

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3.24}$$

Recall measures the proportion of actual positives that have been identified correctly. This is also called the *true positive rate*:

$$\text{Recall or True Positive Rate} = \frac{TP}{TP + FN}. \tag{3.25}$$

Additionally, specificity can be found as the proportion of correct identifications of actual negatives, and the *false positive rate* is then calculated as $1 -$ specificity [46]:

$$\text{Specificity} = \frac{TN}{TN + FP}. \tag{3.26}$$

$$\text{False Positive Rate} = 1 - \text{specificity} = \frac{FP}{FP + TN}. \tag{3.27}$$

During later experimentation on the rehabilitation data set, precision and recall will be measured such that a more thorough comparison can be conducted of the different federated algorithms.

3.3.4 Receiver Operating Characteristics

For binary classifiers, a good way of visualizing prediction performance is by plotting the true positive rate as a function of the false positive rate while varying the classification threshold shown on (3.21). This plot is called a Receiver Operating Characteristic (ROC) curve. An example is shown below in Figure 3.8. [46]

For an ideal classifier the curve should pass through (0,1), meaning that all actual positives have been identified correctly with no false positives. A straight line from (0,0) to (1,1) indicates the performance of a random or *no information* classifier. This means that the classifier does not perform better than a coin toss. [46]

As TPR and FPR change depending on the threshold, selecting a good threshold must depend on the problem at hand. A value of 0.5 is often used as the default threshold.

Area Under the Curve

The overall performance of a model over all thresholds can be expressed as the area under the curve (AUC). A model that classifies randomly, given new data, is expected to have an AUC of 0.5. During experimentation with the rehabilitation data set, AUC will be used in line with the studies described in the state of the art [12, 21, 27]. Throughout the thesis, AUC will be used to specifically refer to the area under the ROC curve.

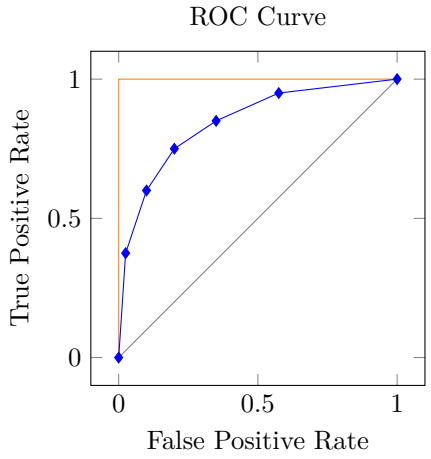


Figure 3.8: Example of several ROC curves. The gray line indicates an AUC of 0.5. The blue curve indicates the performance of a model better than a random guesser. The orange curve is the result of an ideal classifier.

Chapter 4

Federated Learning Theory

In this chapter, some common federated learning approaches will be described along with the algorithms selected for later experimentation. The main advantages and disadvantages of using a federated learning approach will be summarized in the end.

4.1 Definition

The term federated learning was introduced in 2016 by McMahan et al. [24]. Later, Kairouz et al. proposed the following broader definition:

> Federated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective. [9]

The federated learning objective can be described as a minimization of the loss functions across K clients [22, 24]:

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}), \quad \text{where} \quad F(\mathbf{w}) &:= \sum_{k=1}^{K} \frac{n_k}{n} F_k(\mathbf{w}), \\ \text{and} \quad F_k(\mathbf{w}) &:= \frac{1}{n_k} \sum_{i\in\mathcal{P}_k} f_i(\mathbf{w}), \end{aligned} \tag{4.1}$$

where $f_i(\mathbf{w})$ typically is a loss function that gives the loss of a sample $\mathbf{x}_i$ given a label y_i using the weights $\mathbf{w}$. $\mathcal{P}_k$ is the local data (partition) available on client k. $F_k(\mathbf{w})$ is then the average loss of the $\mathcal{P}_k$ samples, where $n_k = |\mathcal{P}_k|$. The objective function $F(\mathbf{w})$ is the weighted average of the average loss $F_k(\mathbf{w})$ of all clients K, where the weighting is depends on the number of local samples n_k. The objective function is then minimized by changing the weights $\mathbf{w}$.

4.2 Federated Learning Settings

As there exists many different kinds of federated learning algorithms, it is necessary to describe the setting of the domain in order to select an appropriate algorithm [16]. Typically, the setting can be categorized as either *Datacenter Distributed*, *Cross-Device*, or *Cross-Silo* [9]. Some of the major characteristics of these settings are described in the following sections.

Datacenter Distributed Learning In this setting, the clients are compute nodes in a single cluster or datacenter. A model is trained on centrally stored data that can be shuffled and balanced between clients. The number of clients varies typically from 1 to 1000, and they are centrally orchestrated. The reliability of the clients is expected to be high. [9]

Cross-Device Federated Learning In the cross-device setting there are up to several billion clients (10^{10}) consisting of mobile or IoT devices, albeit only a fraction of these are available at a given time. The data is generated and stored locally on each client where it is inaccessible for other clients. Model training is orchestrated by a central server with no raw data access. Because there is a very large number of clients, the communication cost is usually the primary bottleneck for training time. [9]

Cross-Silo Federated Learning This setting fits well with the problem domain of this thesis, because the data is siloed in the clients or, in this case, municipalities. Typically, the number of clients vary from 2 to 100, where each client generates and stores their own data locally. Again, the data is unlikely to be exactly similar across clients. The data also *remains* at the clients where it can not be read by other clients. The clients are used for training a model using a central orchestration server, and the orchestrator never sees the raw data of the clients. In contrast to the cross-device setting, the cross-silo clients are expected to have higher reliability, because they should not be limited by e.g. metered network or battery. [9]

4.3 Training Process

A typical training process in federated learning is the following [9]:

1. **Client selection**. Clients are selected based on eligibility requirements. For instance, in the cross-device setting, Google's Gboard for Android devices trains only on devices that are idle, charging, and on a free wireless connection [10]. Again, for cross-silo settings, clients are expected to be more reliable, so that most clients are available for training at a given time.
2. **Broadcast**. The current model weights and training program are sent to the selected clients.
3. **Client computation**. The local client models are updated using the training program, e.g. Stochastic Gradient Descent (SGD).

4. **Aggregation**. The server collects aggregates of device updates. This step may also include additional techniques for e.g. differential privacy which will be described later.

5. **Model update**. The server updates the shared model locally which completes the *round*. The process can then begin again from step 1 - starting a new round.

An illustration of this training process is shown on Figure 4.1, where A) a model is distributed to a number of clients (e.g. municipalities), B) the clients train locally which provides different models that C) are aggregated on a server for D) starting a new round by distributing the new aggregated model to the clients.

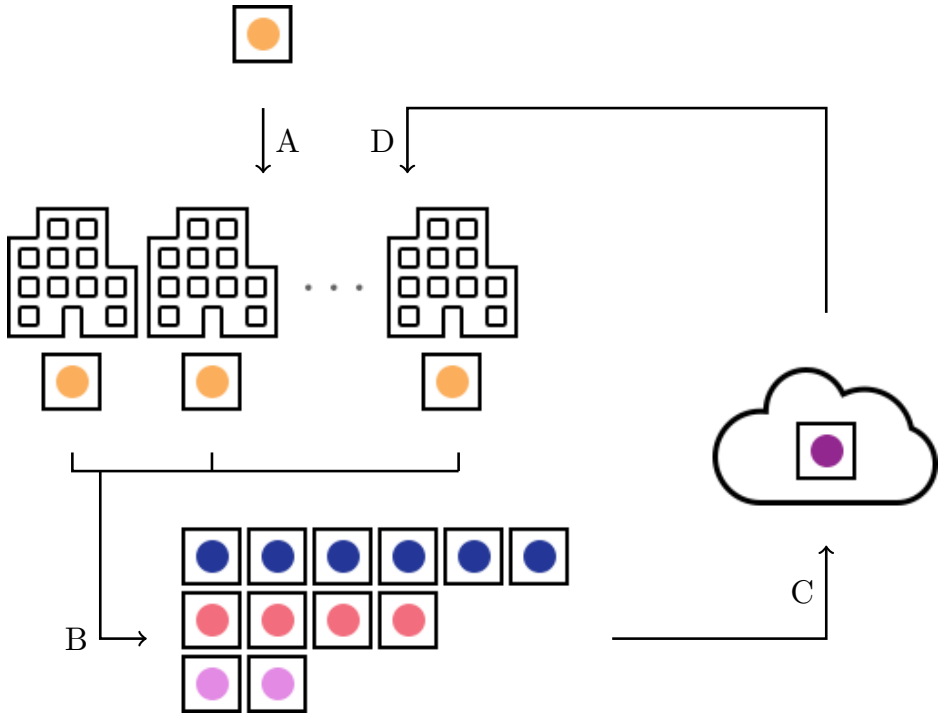


Figure 4.1: Illustration of a typical federated learning process.

4.4 Data Partitioning

A federated learning setting can also be characterized by the data partitioning axis. It is common to use the terms *horizontal* and *vertical* to indicate a partitioning by either samples or features respectively [6]. Different federated learning algorithms have been developed specifically to these types of partitioning [16].

Horizontal Federated Learning This partitioning is also called sample-based federated learning, and it is used in scenarios where datasets share the same feature space but have different sample IDs [6]. For instance, two municipalities working with physiotherapy-based

rehabilitation have data with similar features. However, a citizen is unlikely to be in a rehabilitation program in multiple municipalities, so the amount of shared citizens is very small.

Vertical Federated Learning This partitioning is also called feature-based federated learning, and it is used in scenarios where datasets share the same sample IDs but have different feature spaces [6]. Consider a bank and a e-commerce company located in the same area. Because of their location, they are likely to share a large amount of customers, but their data is different. The bank has data for e.g. income and expense, while the e-commerce company has browsing and purchase history [6].

In the setting of datacenter distributed learning, the data can be partitioned arbitrarily across clients because the data is centralized. For the cross-silo setting, the data can be partitioned either horizontally or vertically [9]. The partition is also fixed, which makes sense because the data should not be moved.

Both horizontal and vertical partitioning is depicted on Figure 4.2.

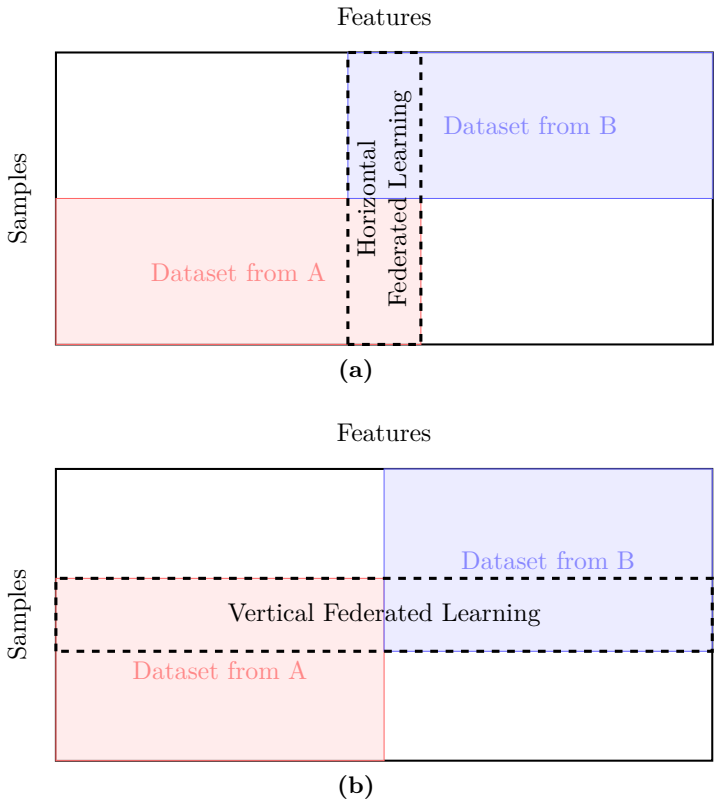


Figure 4.2: Illustration of data partitioning cases in federated learning inspired by Yang et al. [6]. The red and blue areas indicate two different datasets A and B respectively. (a) shows horizontal federated learning where samples are different but features are the same. (b) shows vertical federated learning where samples are the same but features differ.

4.5 Data Distribution

In federated learning there are many challenges to be considered. For instance, improving communication efficiency, algorithm effectiveness, hyperparameter optimization, and protection against malicious actors trying to gain information or tamper with the learning process [9, 17].

Another issue is that data often differs across clients in a real-world federated learning setting. This is often referred to as "non-IID" - not independent and identically distributed. It can be beneficial to use non-IID data for benchmarking the performance of both non-federated data-centralized models and models in a federated setting. Some of the common ways data can be non-IID are described next [9].

In this section we let x represent the features, and y represent the labels of some supervised task. In order to get access to data in a federated learning setting, one must first sample a client $k \sim Q$, where Q is the distribution of available clients. This client's local data distribution is then $\mathcal{P}_k(x, y)$. A sample can then be drawn from this distribution: $(x, y) \sim \mathcal{P}_k(x, y)$. This can be analysed further by rewriting: $\mathcal{P}_k(x, y) = \mathcal{P}_k(x \mid y)\mathcal{P}_k(y) = \mathcal{P}_k(y \mid x)\mathcal{P}_k(x)$.

- **Feature distribution skew (covariate shift)**. When the marginal distribution $\mathcal{P}_k(x)$ changes across clients - even when $\mathcal{P}(y \mid x)$ is the same for all clients, meaning that the same label corresponds to different features. For instance, across municipalities there might be different types of elderly citizens that are all considered able to complete a rehabilitation program. Another example is the differing handwriting of different authors who all write the same letters or digits. The characters corresponding to the same labels will then have differing samples.
- **Label distribution skew (prior probability shift)**. When the label distribution $\mathcal{P}_k(y)$ changes across clients - even when $\mathcal{P}(x \mid y)$ is the same for all clients. For instance, if there is a difference across municipalities, in the number of elderly citizens that would be considered able to complete a rehabilitation program. This can, as an example, be created by taking a data set and partition it by the labels. This would skew the distribution of labels across partitions, since each partition only contains one label that no other partition has.
- **Same labels, different features (concept shift)**. When the conditional distribution $\mathcal{P}_k(x \mid y)$ changes across clients - even when $\mathcal{P}(y)$ is the same for all clients, meaning that the same label can have different features among different clients. For instance, across municipalities there might be a change in the type of elderly citizens that are able to complete a rehabilitation program.
- **Same features, different labels (concept shift)**. When the conditional distribution $\mathcal{P}_k(y \mid x)$ changes across clients - even when $\mathcal{P}(x)$ is the same for all clients. For instance, if a caseworker decides that an elderly citizen is able to complete a rehabilitation program in one municipality, but another caseworker in a different municipality deems the same elderly citizen unable to complete the rehabilitation program.
- **Unbalanced data**. Finally, the amount of data can vary greatly across clients. Some clients may have very little data while others have large amounts of data.

In the coming experimentation part, a benchmarking data set will be used to simulate a non-IID data distribution in a federated setting. The data will be preprocessed to skew the label distribution such that some clients have less of some labels and more of others. Label distribution is selected for experimentation as it is likely that there exists some skew in the amount of elderly citizens that may complete a physiotherapy-based rehabilitation program across municipalities. Furthermore, experimentation will also be conducted with both balanced and unbalanced data across clients, since it is unlikely that all municipalities have a balanced amount of data in the real case. Thus, four cases of data distribution will be experimented with: balanced and unbalanced data with and without label distribution skew.

If it is the case that the real rehabilitation data coming from different municipalities, is correlated with the individual municipalities, one might ask whether a global (shared) model is favourable in the setting of physiotherapy-based rehabilitation. Thus, another experiment will be conducted where accuracy metrics will be determined on each client with respect to the client's local data without using a global model. If a global model does not perform well, it may be beneficial to look into model personalization, where the models on each client adapt specifically to the client it is located on.

4.6 Privacy Preservation for Federated Learning

In a federated learning scenario there is inherently some amount of privacy preservation present because the sensitive data never leaves the client devices, meaning that a data leakage is very unlikely. Although federated learning significantly improves privacy versus a data-centralized approach, federated learning is not perfectly privacy-preserving by itself. As mentioned earlier e.g. training data extraction is still possible if the model memorizes specific samples [13]. It is also possible to perform model inversion attacks that can reconstruct data using the model parameters [14]. A black-box approach, without model parameters, can also reveal information to an adversary through membership attacks [15], that can determine the presence of a particular data record in the training data set.

Privacy is an important aspect in the setting of physiotherapy-based rehabilitation because of its sensitive healthcare data. Therefore, a federated decision support system in this domain must take this into account as well if a such system is to be used in a real-life scenario. In spite of the above, it may be far-fetched to assume that e.g. case workers, other care givers, engineers, and analysts will attempt to perform privacy infringing attacks on the models and the training process to gain access to the sensitive data of elderly citizens.

In the following sections, some threat models will be described along with methods that can preserve privacy against these threat models.

4.6.1 Threat Models

Determining if a system is secure is an undecidable problem [47], so when designing a system it is often a good idea to model potential threats such that mitigations can be prioritized. The

focus of a threat model is to describe who to secure the system against. Table 4.1 shows various threat models based on the work of Kairouz et al. [9].

Access Point/Data	Actor	Description
Client	Someone who has access root access to the client device. E.g. a device administrator or a malicious actor who has gained access in some way.	A client who is able to see messages from the server, but can not interfere with the training process is called honest-but-curious. A malicious client can also tamper with the training process.
Server	Someone who has root access to the server. E.g. a server administrator or a malicious actor who has gained access in some way.	An honest-but-curious server is able to see all messages received, but can not interfere with the training process. A malicious server can also tamper with the training process.
Output Models	Engineers and analysts.	Engineers and analysts may have access to different outputs from the system. For instance, models from multiple training rounds.
Deployed Models	Any municipality participating, or possibly the rest of the world.	The final model may be deployed to any municipality in Denmark. If a model leakage occurs, it may provide either black box or white box access to the rest of the world.

Table 4.1: Various threat models for federated learning.

The current setting of physiotherapy-based rehabilitation is characterized by a large degree of trust between municipalities. Therefore, it is assumed that municipalities are at least honest-but-curious, meaning that it is possible for sent messages to be inspected, but that the training process will not be interfered with. Moreover, it is also assumed that participants are non-colluding, meaning that two or more participants are not conspiring against other participants to learn of their sensitive data of elderly citizens.

4.6.2 Data Anonymization

A popular approach to privacy, used at e.g. Google [48], is k-anonymity [49]. Given a table where the rows represent different people and the columns represent different attributes, a data publisher can then anonymize the data by removing certain values (suppression), or replace certain values with a broader category (generalization) [49]. For instance, binning the birth year 1974 into decade 1970. In k-anonymity, quasi-identifiers are the attributes available to an adversary that can not be used as unique identifiers. A k-anonymous data set guarantees that tuples of quasi-identifiers occur at least k times in the data set [28], which makes it more

difficult for an adversary to identify a specific individual. For instance, a table consisting of quasi-identifiers: gender, age, and municipality has 2-anonymity if there are at least two rows with the same attributes, for any combination of the attributes in any row.

An example of anonymization is the well-known competition by Netflix in 2006, where the goal was to improve their movie recommendation service. Netflix provided an anonymized data set containing more than 100 million movie ratings created by about 500 thousand users. It was later shown by Narayanan et al. [28] that de-anonymization is possible using a small amount of auxiliary data. Due to the uniqueness of the rows, or users, in the data set, the authors were able to develop an algorithm to effectively determine matching identities using information from IMDb - the Internet Movie Database. They showed that 84% of users can be identified if the adversary has knowledge of six out of eight rated movies outside of the top 500 blockbusters.

Currently, the most prevalent methods for providing privacy guarantees for federated learning is through differential privacy and cryptographic methods such as Secure Multi-Party Computation (SMPC), and homomorphic encryption [9, 16, 17]. In the coming sections differential privacy and SMPC will be described.

4.6.3 Differential Privacy

Differential privacy is described, by Dwork et al. [50, ch. 1], as a promise from a data holder, or curator, to a data subject. The promise is that data subjects are not affected, adversely or otherwise, by allowing their data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available. Differential privacy ensures that the participation of e.g. a survey in itself, is not revealed [50, ch. 2], however the participant can of course still be affected by the result of the survey.

Privacy is preserved under *post-processing*, which means that even if an adversary has a large amount of computing power and other information sources available (auxiliary data), privacy will not decrease by processing differentially private output. Consequently, it is possible for a differentially private database to be made publicly available without compromising privacy and resorting to e.g. data usage agreements.

The differentially private methods work by introducing randomness, or noise, such that information of a specific individual is not revealed in a way that compromises the individuals privacy [50, ch. 11]. A sufficient amount of noise should be added to the data such that privacy is preserved, however, to also gain utility from the data, the amount of noise should be minimized. Here, "gaining utility" means we are able to learn the statistical information we are interested in by using the data. Therefore, a trade-off exists between gaining utility and protecting privacy. As previously mentioned the goal in training a machine learning model is to achieve a model that is able to generalize to unseen data. The constraint of privacy does not conflict with this goal, as we are not interested in learning about individuals, but rather reveal general distributional information [50, ch. 11].

Randomized response

An example of how the introduction of randomness can provide privacy, is randomized response. Assume that a social scientist would like to know how many people have committed murder. Participants in such a study would likely be reluctant to answer truthfully if their answer is incriminating. The scientist would then ask participants to flip a coin, and if the result is heads, answer truthfully. Otherwise, flip the coin again. Now, if the coin is heads, answer "Yes", and "No" otherwise.

Due to the coin-flipping the number of murderers answering "Yes" is expected to be $\frac{3}{4}$. For non-murderers the value is $\frac{1}{4}$. Thus, if p is the true fraction of participants that committed murder, then the expected number of "Yes" answers is then $y = \frac{3}{4}p + \frac{1}{4}(1-p) = \frac{1}{4} + \frac{p}{2}$. The value of p can then be estimated by $p = 2y - \frac{1}{2}$. Due to the introduction of randomness any outcome can be plausibly denied, resulting in privacy-preservation [50]. It is however still possible to estimate p, the true fraction of murderers among the participants, given that enough people participate in the study.

(ϵ, δ)-differential privacy

In differential privacy, it is assumed that data from individuals are held in a database D with n rows, where each row corresponds to an individual person [50, ch. 2]. The definition of (ϵ, δ)-differential privacy by Dwork et al. is the following: let D and D' be any adjacent databases, meaning that they differ in only one record, i.e. one of the databases has a record more than the other. Then, a randomized mechanism $\mathcal{M}$ with domain $\mathcal{D}$, the set of all possible databases, and range $\mathcal{R}$ is (ϵ, δ)-differentially private if for all $\mathcal{S} \subseteq \mathcal{R}$ [50, ch. 2]:

$$\Pr[\mathcal{M}(D) \in \mathcal{S}] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(D') \in \mathcal{S}] + \delta. \tag{4.2}$$

M is a *mechanism*, which means that it represents any interaction with the data D and D', such as an algorithm or function. In our case this is a model training algorithm. Additionally, it is *randomized* because it uses noise. Differential privacy requires that the probability distributions over outcomes of the mechanism are similar for two adjacent data sets. In a slightly more intuitive fashion, this means that the outcome of the mechanism is not dependent on the inclusion of a specific record, e.g. the data of a user. Equation (4.2) guarantees that the privacy loss of two adjacent databases D and D' is bounded by ϵ with a probability of at least $1 - \delta$. The probability δ is the risk of breaking the differential privacy guarantee, where the value is preferably smaller than the inverse of the length of the database $\frac{1}{|D|}$ [50, ch. 2]. The ϵ bounds the privacy loss and is commonly referred to as the privacy budget. A higher value of ϵ means less privacy and a lower value means more privacy. If $\delta = 0$ then $\mathcal{M}$ is said to be ϵ-differentially private. If $\delta = 0$ and $\epsilon = 0$, then the probability distributions are equal and $\mathcal{M}$ is perfectly private.

A question that may arise is: how much noise is necessary to add through the mechanism? Optimally, the noise should be high enough to provide the privacy guarantees, and also low enough to ensure data utility such that an accurate model can be trained.

Sensitivity

A common pattern of providing differential privacy guarantees is to first calculate the *sensitivity* of the function, or query, we would like to apply to the database, and then calibrate how much noise to add based on this quantity [32]. The query could be the question an analyst would like to know, e.g. calculating a sum over a database [51]. Here a real-valued function $f : \mathcal{D} \to \mathbb{R}$ is used as an example. The sensitivity is defined as the maximum possible amount of change in the output of function f over adjacent databases:

$$\Delta f = \max \left\| f(D) - f(D') \right\|_1, \tag{4.3}$$

where D and D' are adjacent databases [50, ch. 3]. The idea is to hide the maximum, or worst-case, possible change in output such that any single record is protected.

Noise

As previously mentioned, noise can help increase privacy, but to allow the data to be utilized through analysis, the noise should not be added excessively. The amount of noise needed to provide privacy is based on three factors [51]:

- Noise type.
- Query sensitivity.
- The desired ϵ and δ.

There are commonly two types of noise that are used for differential privacy: Laplace noise and Gaussian noise. The Laplace mechanism is based on Laplace noise with a mean of 0, and is defined as:

$$\mathcal{M}_L(D) = f(D) + \text{Lap}(\Delta f/\epsilon), \tag{4.4}$$

where $\text{Lap}(\Delta f/\epsilon)$ denotes a random variable drawn from the Laplace distribution with the following probability density function where the scale $b = \Delta f/\epsilon$:

$$\text{Lap}(x \mid b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right). \tag{4.5}$$

If the noise is added to a function that returns real values, and if the noise is drawn from $\text{Lap}(\Delta f/\epsilon)$, then it can be shown that $(\epsilon, 0)$-differential privacy is preserved [50, ch. 3]. Another mechanism is the Gaussian mechanism used by Abadi et al. [32] to create a differentially private version of the stochastic gradient descent algorithm. The Gaussian mechanism is defined as [32]:

$$\mathcal{M}_G(D) = f(D) + \mathcal{N}(0, \Delta f^2 \sigma^2), \tag{4.6}$$

where $\mathcal{N}(0, \Delta f^2 \sigma^2)$, is a Gaussian distribution with mean 0 and standard deviation $\sigma \Delta f$.

Composability

To create more sophisticated algorithms it would be beneficial to have the composition of differentially private algorithms to also be differentially private. For k ϵ-differentially private mechanisms each applied on a disjoint subset of the private database, the ultimate privacy guarantee is bounded by the worst guarantee of the mechanisms, and not the sum [52]. I.e. $(\max_i^k \epsilon_i)$-differential privacy is guaranteed.

It has been shown that the sequential composition of k (ϵ, δ)-differentially private mechanisms is at least $(k\epsilon, k\delta)$-differentially private [50, ch. 3]. Other tighter bounds have been found, however, these will not be described here.

Local and Global Differential Privacy

Local differential privacy, or distributed perturbation [53], adds the noise to individual data samples (mechanism input). For instance, randomized response is an example of local differential privacy since each participant is responsible for adding noise before the results are handed to the scientist for analysis. In federated learning, local differential privacy can be achieved by letting clients apply differentially private perturbations before sharing data, e.g. model updates, with the central server. The clients are most protected in this manner because they do not rely on any external parties for privacy protection. [9]

Global differential privacy, or centralized perturbation [53], adds noise to the outputs after model aggregation by the central server. In the federated setting this requires a trusted server because the noise is added at the server side. The benefit of this approach is that less noise may be needed, which increases the utility of the data but still provides the same privacy guarantees as local differential privacy.

Summary of Benefits and Drawbacks

Differential privacy can protect against arbitrary threats and guarantees that privacy is preserved under post-processing. For instance, privacy is preserved even if auxiliary data is available and adversaries have unbounded computational capabilities. The privacy is guaranteed by adding noise, however, the added noise likely leads to decreased utility. A benefit of differential privacy is that the privacy loss bound ϵ and risk of disclosure δ allows comparison of different private mechanisms. A drawback is that continuous querying allows the ground truth to be estimated. As an example, drawing a sample from a univariate Gaussian distribution does not say much about the distribution of which it was sampled. However, continuous sampling can eventually reveal the mean and variance. Overly accurate answers to too many questions will destroy privacy, and thus the goal of algorithmic research on differential privacy is to postpone this inevitability as long as possible [50].

Local differential privacy can protect privacy against an untrusted central server. This is the most private setup because privacy is still preserved even if the model is leaked after leaving the clients. Global differential privacy can provide better accuracy at the same level of privacy as local differential privacy, but it requires a trusted server.

4.6.4 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) is a subfield of cryptography that focuses on allowing multiple parties to calculate a function on their own private input without disclosing the inputs or intermediate results. This can be of relevance in federated learning because the confidentiality of model updates can be protected - allowing only the aggregation of updates to be revealed. [9, 51] This is partly possible because the central aggregator should not be interested in the model updates of individual clients, but rather the sum, or average. When using SMPC in this manner, there is no need for a trusted aggregator.

Additive Secret Sharing

Additive secret sharing refers to cryptographic methods that can be used to split a secret number into shares such that the sum of the shares equals the secret number. I.e. for a secret a split into n uniform additive secret shares $a_i \in \mathbb{F}_p$ [54]:

$$a = a_1 + ... + a_n, \tag{4.7}$$

where also $a \in \mathbb{F}_p$. Computation is typically done over a finite field $\mathbb{F}_p$ of characteristic p [54], meaning that computation happens over a finite set of numbers: $0, 1, ..., p-1$, where p is a large prime [51]. The values a_i are uniformly chosen at random in $\mathbb{F}_p$. A data item $a \in \mathbb{F}_p$ is $\langle\cdot\rangle$-shared if a player, or client, P_i holds the value a_i. This can be used to protect the model updates from the central server that aggregates these updates, since aggregation can be performed in an encrypted state, meaning no client or server can see what the true gradients are [51]. Because additive secret sharings are linear, addition can be performed locally in an encrypted state [55]:

$$\langle x \rangle + \langle y \rangle = (x_1 + y_1, ..., x_n + y_n) = \langle x + y \rangle . \tag{4.8}$$

This property is used during experimentation of SMPC where the updated model weights from the clients are shared between all clients before aggregation. It is possible to encrypt, not only the model updates, but also the model itself such that the model is not leaked to clients. To calculate a forward pass in a neural network it must be possible to do multiplication with secret shares. In order to compute $\langle x \cdot y \rangle$ given $\langle x \rangle$ and $\langle y \rangle$, the clients need to have so-called multiplication triples $\langle a \rangle$, $\langle b \rangle$, $\langle a \cdot b \rangle$ where $a, b \in \mathbb{F}_p$. Then the product $\langle x \cdot y \rangle$ can be calculated as [54, 55]:

$$\langle z \rangle \leftarrow \langle a \cdot b \rangle + \delta \cdot \langle a \rangle + \varepsilon \cdot \langle b \rangle + \delta \cdot \varepsilon, \tag{4.9}$$

where $\varepsilon \leftarrow \langle x - a \rangle$ and $\delta \leftarrow \langle y - b \rangle$. We note that the δ here does not have anything to do with the constant in differential privacy. To get the values of ε and δ they must be *reconstructed*, meaning that the clients must coordinate to add their shares. Since we do not experiment with encrypted models, we will not go into further detail.

Shamir Secret Sharing

The additive secret sharing scheme described earlier does not reveal a given secret without the consent of all n clients, however, this may be an issue if clients drop out during the fed-

erated learning process. Another method of secret sharing is Shamir secret sharing where the secret can be reconstructed even if some shareholders are unavailable [56]. Given a secret x, and a random polynomial function f where $f(0) = x$, the secret can be split into n shares: $f(1), f(2), ..., f(n)$ that are points in a 2-dimensional plane. The number of sufficient shares needed for reconstruction can be adjusted by changing the degree of f. Specifically, the polynomial must be of degree $T - 1$ if T is the sufficient amount.

Fixed Precision Encoding

Since the model weights in a neural network are often real numbers they must be encoded using integers to be additively secret shared. A way of doing this, is to scale the weights with a constant s using:

$$\tilde{x} = \text{encode}(x) = x \cdot s \mod p, \tag{4.10}$$

where

$$s = \text{base}^{\text{precision}}, \tag{4.11}$$

and p is the large prime from earlier. For instance, if the base is 10 and the precision is 3, then $s = 10^3$ and thus a rational number 1.234 can be encoded as 1234. Decoding can be done using:

$$x = \text{decode}(\tilde{x}) = \begin{cases} \tilde{x}/s, & \text{if } \tilde{x} \leq p/2 \\ (\tilde{x} - p)/s, & \text{otherwise} \end{cases}, \tag{4.12}$$

where numbers $\tilde{x} \leq p/2$ are decoded as positive numbers, and numbers $\tilde{x} > p/2$ are decoded as negative numbers. [51]

Summary of Benefits and Drawbacks

The use of SMPC generally requires significantly more computation power and also adds communication overhead, but the private model updates can be protected even if a substantial amount of clients $(n-1)$ are malicious and colluding. The processing of model update aggregation can happen in an encrypted state, and privacy is preserved even against adversaries with powerful computational capabilities. Using SMPC during model update aggregation should have virtually no impact on model quality.

Drawbacks include increased federated training time because of secret shares that need to be distributed between clients. A disadvantage of additive secret sharing is that the number of shares, that a secret is split into, is also the minimum number of shares needed to reconstruct the secret. Therefore, lack of robustness may be an issue with unreliable shareholders. In a cross-silo federated setting it is a reasonable assumption that clients are adequately reliable.

4.7 Federated Learning Algorithms

In this section several federated learning algorithms will be presented. The algorithms that will be used later during experimentation are: Federated Averaging, Federated Averaging with Differentially Private Stochastic Gradient Descent, and Federated Averaging with SMPC.

4.7.1 Federated Averaging

Federated Averaging is a popular federated learning algorithm proposed by McMahan et al. that is resistant to unbalanced and non-IID datasets [24]. The algorithm works by letting clients apply several mini-batch SGD updates locally, and then sharing the updated model weights with a central server which then aggregates all client updates and redistributes the final weights.

In alg. 1, the algorithm first initializes a global model with parameters $\mathbf{w}_0$. Afterwards, the server selects a fraction, C, of K total number of clients to participate in a given round t. If $C = 1$ then it indicates that all clients are participating.

In every round each client k performs an update using the current model $\mathbf{w}_t$ from the server. A client update consists of first splitting the local data partition $\mathcal{P}_k$ into batches of size B. Then, for each batch, the client updates the local weights by calculating the gradient of the loss function L using backpropagation. All batches are processed locally E times (local epochs). Finally, the server aggregates the weights of all clients using weighted averaging, where the weighting is defined by the fraction of samples on the specific client $n_k = |\mathcal{P}_k|$ with respect to the total number of samples n. I.e. $\mathbf{w}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{t+1}^k$. For non-convex problems, a disadvantage of this approach is that averaging models in parameter space may result in an arbitrarily poor model [24, 31]. Despite this, McMahan et al. [24] find that the naive parameter averaging works surprisingly well in practice when the models start from the same random initialization $\mathbf{w}_0$.

By increasing client participation C, the communication with the server increases, meaning that the computational load of the server increases, because the server must aggregate over a larger number of clients. Modifying epochs E and batch size B affects client computation load. When $C = 1$, $E = 1$, and $B = \infty$, then the algorithm performs only a single gradient descent step on each client every round. This is referred to as Federated SGD [24].

4.7.2 Federated Averaging with Differentially Private Stochastic Gradient Descent

Inspired by Zhao et al. [53], we implement local differential privacy by perturbing the model updates from the clients. However, instead of applying this to Federated SGD, we here try to use Federated Averaging. In alg. 2 the Federated Averaging algorithm has been modified to include the differentially private stochastic gradient descent algorithm from Abadi et al. [32]. The server functionality is the same as for Federated Averaging, but the client update function is now different. This version of SGD limits the amount of information that a single client can contribute with during each round. Abadi et al. uses the term *lot* for describing the group of samples of which the gradient of the loss function is calculated and thereafter averaged (i.e. SGD mini-batch). In the following section, this group of samples is renamed to *batch*.

For every step t of the total number of steps T, the client begins by first sampling a batch where each sample is independently picked with probability $q = B/n$ where B is batch size and n is the total number of samples. In practice this can be done by permuting the samples and then partitioning into appropriate sizes. During experiments the value of T is set to

Algorithm 1: `Federated Averaging`.

Function `Server()`**:**
initialize $\mathbf{w}_0$
for *each round t = 1,2,...* **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**
$\mathbf{w}_{t+1}^k \leftarrow$ `ClientUpdate(`k, $\mathbf{w}_t$`)`
$\mathbf{w}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{t+1}^k$

Function `ClientUpdate(`k, $\mathbf{w}$`)`**:**
$\mathcal{B} \leftarrow$ (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**
$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; b)$
return $\mathbf{w}$ to server

$T = E/q$ [32], which corresponds to the total number of batches $1/q$ for E epochs.

Then, before applying the gradient to the weights immediately after computing it, the gradient is preprocessed by first clipping it and then by adding noise. The gradient clipping is done using the ℓ_2-norm where the clipping threshold is set to V, meaning that when $\|\mathbf{g}(\mathbf{x}_i)\|_2 > V$, then $\mathbf{g}(\mathbf{x}_i)$ is scaled to a norm of V. If $\|\mathbf{g}(\mathbf{x}_i)\|_2 \leq V$, then $\mathbf{g}(\mathbf{x}_i)$ is not scaled, so the gradient is preserved. The reason why the gradient is clipped, is because the size of the gradient is not known *a priori* [32]. This is important, because after clipping the gradient, noise is added to provide differential privacy guarantees where the noise is scaled based on the clipping threshold. The noise is normally distributed and it is added to the sum of gradient vectors $\sum_i \overline{\mathbf{g}}(\mathbf{x}_i)$ with a mean of zero. The standard deviation of the noise distribution is σV. The idea here is to scale the noise with the maximum clipping threshold V such that any gradient norm less than this are protected as well.

To keep track of the overall privacy budget ϵ, Abadi et al. suggests a "moments accountant" that relies on the composability of differential privacy. If every access to the gradient during training is differentially private, then the resulting model will also be differentially private. The authors prove that their modified SGD algorithm is $(O(q\epsilon\sqrt{T}), \delta)$-differentially private, meaning that decreasing batch size B, increasing the size of the data set n, or decreasing the total number of steps T will yield a lower privacy budget – providing better privacy guarantees. During experimentation this differentially private SGD algorithm is implemented using Opacus that tracks the privacy budget internally using Rényi Differential Privacy but reports results in the (ϵ, δ) language.

Algorithm 2: `Federated Averaging with Differentially Private SGD (outline).`

Function `Server()`**:**

initialize $\mathbf{w}_0$
for *each round* $t = 1, 2, ...$ **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**
    $\mathbf{w}_{t+1}^k \leftarrow$ `ClientUpdate(`k, $\mathbf{w}_t$`)`
  $\mathbf{w}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{t+1}^k$

Function `ClientUpdate(`k, $\mathbf{w}$`)`**:**

for *each step* t *from* 1 *to* T **do**
  Take a random sample b with sampling probability B/N
  Compute gradient
  For each $i \in b$, compute $\mathbf{g}(\mathbf{x}_i) \leftarrow \nabla_{\mathbf{w}} L(\mathbf{w}, \mathbf{x}_i)$
  Clip gradient
  $\overline{\mathbf{g}}(\mathbf{x}_i) \leftarrow \mathbf{g}(\mathbf{x}_i) / \max(1, \frac{\|\mathbf{g}(\mathbf{x}_i)\|_2}{V})$
  Add noise
  $\tilde{\mathbf{g}} \leftarrow \frac{1}{B} \left(\sum_i \overline{\mathbf{g}}(\mathbf{x}_i) + \mathcal{N}(0, \sigma^2 V^2 \mathbf{I}) \right)$
  Descent
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \tilde{\mathbf{g}}$
Compute privacy cost (ϵ, δ) using a privacy accounting method.
return $\mathbf{w}$ to server

4.7.3 Federated Averaging with SMPC

For the coming experiments with SMPC, the Federated Averaging algorithm is modified such that the clients now encrypt the model weights before returning these to the central server. It is here assumed that the server is honest-but-curious, meaning that the server follows the protocol but may see all messages received. The algorithm is shown in alg. 3 where mostly the client update function has been updated. After updating the local model weights, the weights are encoded using fixed precision such that the real numbers can be additively secret shared with other clients. Then, the shares received of other participating clients are summed and returned to the server. After the server receives these weights and performs aggregation, then the "true" weights appear due to reconstruction. Finally, the weights are fixed precision decoded and averaged before starting a new round.

4.8 Explainability

As mentioned previously, in a real-life clinical scenario, the users of a decision support system must be able to trust the model and its predictions to allow the predictions to be used in

Algorithm 3: `Federated Averaging with SMPC (outline).`

Function `Server()`**:**
- initialize $\mathbf{w}_0$
- **for** *each round t = 1,2,...* **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**
    - $\mathbf{w}_{t+1}^k \leftarrow$ `ClientUpdate(`k, $\mathbf{w}_t$`)`
  - $\overline{\mathbf{w}}_{t+1} = \text{FixedPrecisionDecode}(\sum_{k=1}^{K} \mathbf{w}_{t+1}^k)$
  - $\mathbf{w}_{t+1} \leftarrow \frac{1}{n}\overline{\mathbf{w}}_{t+1}$

Function `ClientUpdate(`k, $\mathbf{w}$, S_t`)`**:**
- $\mathcal{B} \leftarrow$ (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**
    - $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; b)$
- $\overline{\mathbf{w}} = \text{FixedPrecisionEncode}(n_k \mathbf{w})$
- $\text{Share}(\overline{\mathbf{w}}, S_t)$
- $\tilde{\mathbf{w}} = \sum_{k \in S_t} \text{ReceiveShares(k)}$
- **return** $\tilde{\mathbf{w}}$ to server

practice. A way of increasing user trust is to explain to the user what the model bases its predictions on. Due to the importance of explainability in clinical settings such as the case with physiotherapy-based rehabilitation, it is relevant to also take this into account when experimenting with the different federated learning techniques. Specifically, it would be interesting to see how significantly the explainability changes due to different learning methods.

4.8.1 SHAP

Achieving a good explanation can be done by using a simple model that can be easily interpreted using the model itself – e.g. one can use a simple linear model or a decision tree. Though, for more complex models such as neural networks it can be exceedingly difficult to understand how a prediction was made, given only the model. In this case, one can use methods such as SHapley Additive exPlanations, or SHAP, to create an *explanation model* which is defined as any interpretable model that can estimate the complex model locally [29, 57]. This means that the explanation model can explain a prediction $f(x)$ based on a single sample x.

SHAP is a framework for calculating feature importances of a given prediction, meaning that each feature is given a score based on how predictive the feature is of a given prediction such that more important features are given higher scores. The authors of SHAP, Lundberg et al. [29], define a class of *additive feature attribution methods* that allows generalization of six

explanation methods including e.g. LIME. These additive methods have the following form:

$$g(\mathbf{z}') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i', \tag{4.13}$$

where g is the explanation model of a more complex model f, and $\phi_i \in \mathbb{R}$, where the ϕ_i's represent feature attribution values. The input to g is a vector $\mathbf{z}' \in \{0,1\}^M$, i.e. $\mathbf{z}'$ is a vector of size M consisting of only ones and zeros that typically represents features being observed or not. The value of $\phi_0 = f(h_x(\mathbf{0}))$, where h_x is a function that maps from the binary values of $\mathbf{z}'$ to the original function input space. The value of ϕ_0 is the base value that would be output if no features were known.

The feature attributions, or SHAP values, are based on the Shapley regression values, that can then be calculated as:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S\cup\{i\}}(\mathbf{x}_{S\cup\{i\}}) - f_S(\mathbf{x}_S)], \tag{4.14}$$

where S is a subset of the input features F. Here, $|S|$ and $|F|$ are the number of features in the respective sets. The second factor on the right hand side, $f_{S\cup\{i\}}(\mathbf{x}_{S\cup\{i\}}) - f_S(\mathbf{x}_S)$, is the difference between predictions of two models trained with and without feature i respectively, where $\mathbf{x}_S$ represents the values of the features in set S. This method requires that the model f is trained on all subsets $S \subseteq F$ because the importance of a feature also depends on the other features available to the model. The summation and the first factor is a weighted average of all the possible differences. The SHAP values are then calculated by defining $f_S(\mathbf{x}_S) = E(f(\mathbf{x}) \mid \mathbf{x}_S)$, where the right hand side is the expected value of the model f given a subset of the input features S. A downside of using Shaply regression values is the computation time that scales exponentially with the number of features – $2^{|F|}$ [29], however there exist methods to approximate these values which will not be reviewed here.

Decision Plot

To make the SHAP values more comprehensible, one can visualize the values using a decision plot as shown in Figure 4.3.

The decision plot shows the most important features in decreasing order along the y-axis for all predictions plotted. The model's output is shown on the x-axis where the plot is centered on the model's expected value ϕ_0 by default. Beginning at the bottom and going to the top, each SHAP value contribution is added to the expected value, or base value. Each line corresponds to the prediction of one sample where the intersection with the x-axis at the top is the predicted value from the model. This predicted value gives the color of the line.

Image Plot

An image plot, shown in Figure 4.4, is another type of plot that can be used to display SHAP values for image data.

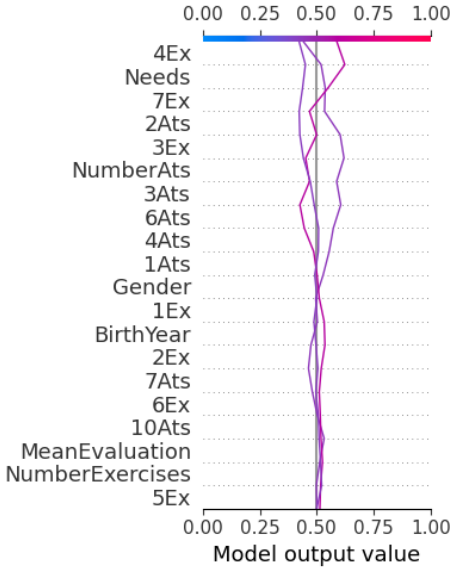


Figure 4.3: An example of a SHAP decision plot showing the most important features in decreasing order along the y-axis for three predictions.

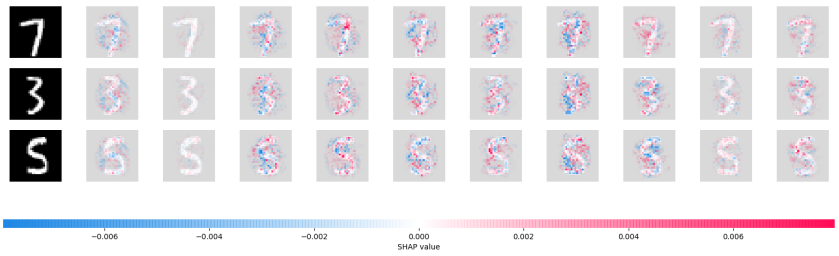


Figure 4.4: An example of a SHAP image plot showing the most important features of three samples along the y-axis for all possible labels along the x-axis.

In this case the image plot shows, to the left, three different image samples of the numbers 7, 3, and 5. The 10 columns of images to the right side correspond to 10 possible labels from zero to nine (left to right). The three samples are used as backdrops, where the SHAP values have been plotted on top. If a feature is important for a specific label it is plotted using a red color, and blue otherwise. Thus, with a good classifier the SHAP image plot should color the numbers red, where the label corresponds to the sample.

4.9 Summary of Benefits and Drawbacks

Some of the main benefits using federated learning are:

- Privacy by design, because sensitive data is not directly shared.

- May allow more data to be used for training because of improved privacy.
- On-site inference reduces latency of model feedback.
- Many privacy mechanisms are not mutually exclusive and they can coexist in a federated learning setting [16]. For instance, differential privacy and SMPC can be used in conjunction.
- Model personalization is possible such that each client has a model that is tailored to them.
- If data-centralization becomes infeasible due to large amounts of data then federated learning can alleviate this problem.

Some of the drawbacks and challenges of using federated learning are:

- Requires client computation.
- May require a lot of client communication although this is less significant in a cross-silo setting compared to a cross-device setting.
- Longer wall-clock training time because of communication delay and client computation time.
- Poor data distribution across clients can affect global model performance negatively.
- Information leakage is still possible by sharing model updates such as weight parameters or gradients, so additional privacy mechanisms may be needed.
- Node failures can affect the globally trained model negatively due to a reduced amount of data.
- Node failures can increase wall-clock training time if the common lockstep training approach is used.
- Techniques such as feature engineering may be challenging since the training data is hidden.
- Hyperparameter optimization can be more resource intensive.
- A model trained in a federated manner may be vulnerable to client data with low quality.

Chapter 5

Design

In this chapter, the design of the experiments and the experimental setup is described.

5.1 The Federated Setting

The experiments are conducted in a cross-silo horizontal federated learning setting, i.e. where only a few clients are present and the clients have data with the same features. This is to increase similarity with the case of physiotherapy-based rehabilitation. For all experiments the proportion of clients available per round C is fixed to 100%, i.e. $C = 1$, where each client has perfect reliability. Furthermore, the datasets for each client are fixed in the sense that neither the sizes nor samples change during training.

5.2 Experiment Process

The process used during the experiments with the federated algorithms can be described as the following procedure:

1. **Load data**. The data set is loaded into memory.
2. **Data preprocessing**. Two data sets are experimented with: MNIST and real rehabilitation data. For MNIST the data set is normalized such that the mean is zero and standard deviation is one. For the real rehabilitation data, only the first eight columns of data are normalized because the remaining columns are feature embeddings.
3. **Data Distribution & Partitioning**. The data set is processed such that the data becomes balanced or unbalanced, with or without label distribution skew across clients. The data is also partitioned into K parts, where K is the number of clients. This provides one part for each client. This is described in more detail in the coming section.
4. **Subpartition**. The K partitions are each randomized and then partitioned into 80% training data and 20% test data.

5. **Train & Test**. A federated algorithm is run for 50 rounds. Each round all clients test their local model on their own local data.

The non-federated model trained on centralized data is given all the available data. In this case there are no communication rounds as in the federated setting, however, the number of epochs is therefore set to 50 to allow comparison. The learning rate η is fixed at 0.01 for all MNIST experiments as this seems to perform reasonably well for the data-centralized case using MNIST. The learning rate for the experiments with the real rehabilitation data is also set to 0.01, because this value is currently used in the AIR project.

In general the federated algorithms are evaluated by varying local epochs E on each client, batch size B, and number of clients K, because these parameters have a great impact on performance. Other interesting parameters that could be experimented with include learning rate, momentum-based gradient descent, and neural network architecture.

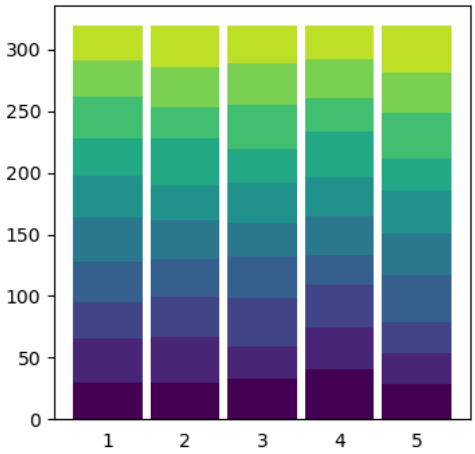
5.3 Data Distribution

The data is preprocessed in such a way that the data across clients is either balanced or unbalanced, with or without label distribution skew. This gives four different combinations that will be used during experimentation. These cases are depicted in Figure 5.1, and the partitioning strategies are as follows:

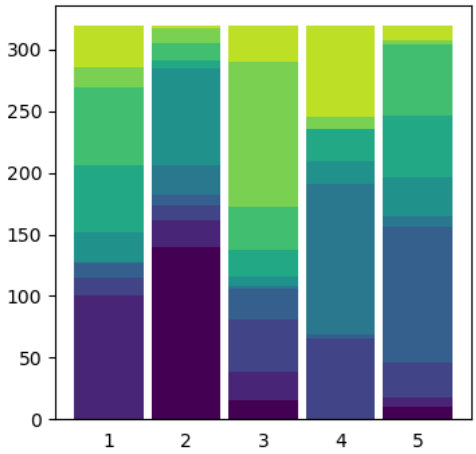
- **Balanced data without label distribution skew**. Randomly permute the order of samples and then make an even partitioning into K chunks, where K is the number of clients.
- **Unbalanced data without label distribution skew**. Randomly permute the order of samples and then randomly partition into K chunks.
- **Unbalanced data with label distribution skew**. The same procedure as Yurochkin et al. [25] is followed for this case. For every class in the data set, K values are drawn from the Dirichlet distribution using a concentration parameter value $\alpha = 0.5$. The data set is then partitioned using these values as proportions. This results in strong skew where it is possible that some clients have none of certain classes. As this is a problem for binary classification on the rehabilitation data set, this constant is changed to $\alpha = 0.95$ for that data set, such that the label distribution is more even. This allows each client to have more samples of each label during both training and testing.
- **Balanced data with label distribution skew**. The same procedure as the unbalanced data case with label distribution skew is followed here. However, the data is subsequently evened out by redistributing samples from the clients that have greater than average amounts of samples to the other clients.

In the experiments of unbalanced data, all clients are allowed to have at least 8% of the data, and at least one of the clients must have at most 10% of the data. This is done because

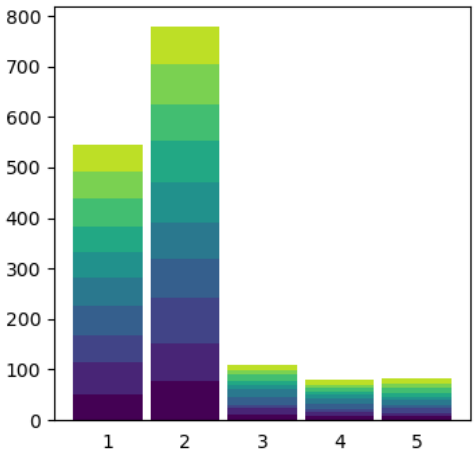
the clients test their model on their own data partition. Having too small data partitions could give inaccurate results. For the MNIST data set used here, this means the minimum partition size is 160 samples for all clients, and maximum 200 samples for one client. The minimum for the real data set is 84 samples for all clients, and maximum for one client is max 104 samples. We note that these values are calculated for $K = 10$ which is the maximum number of clients that are experimented with.



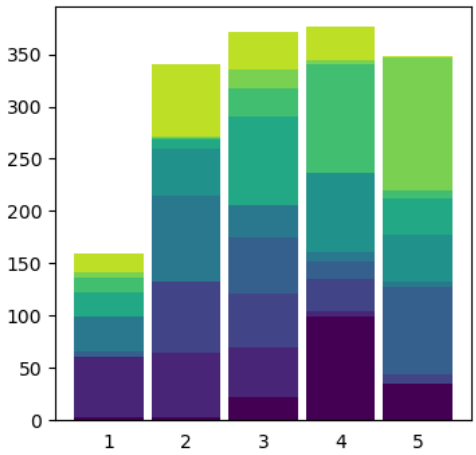
(a) Balanced data with no (very little) label distribution skew.



(b) Balanced data with label distribution skew.



(c) Unbalanced data with no label distribution skew.



(d) Unbalanced data with label distribution skew.

Figure 5.1: These diagrams show different data distribution scenarios across $K = 5$ different clients. The first axis shows the clients from 1 to 5, and the second axis shows the total number of samples for these clients. The different color gradients correspond to different labels.

5.4 Measuring Computation Time

The computation time is measured by saving the timestamp at both the start and the end of the training periods. The amount of work unrelated to training is minimized during those two timestamps. Furthermore, the computing specifications are fixed during all experiments. An Intel i5-6600K @3.50 GHz processor is used along with 16 GB memory. A GPU was not used during experimentation.

5.5 Neural Network Architectures

On the MNIST dataset the dimensions of the neural network model are: 784, 128, 128, 10, where each number indicates the number of neurons. Thus, there are 2 hidden layers of size 128, and an output layer of 10 – one for each class. The MNIST images are vectorized, or flattened, such that they can propagate through the network. I.e. each image of size 28 by 28 pixels becomes a single vector of size $28 \cdot 28 = 784$. Hence, the input layer size.

For the rehabilitation data the neural network model dimensions are: 27, 64, 2. For binary classification it suffices to have only a single output neuron, which results in a lower amount of parameters to train. In this case the output layer size of 2 is chosen because of implementation convenience. The dropout rate is set to 0.5 for both models which is also the recommended value for most tasks [44]. The activation function is set to ReLU for all layers except the last. For the last layer, the activation function is softmax. This model for the rehabilitation data is similar to the neural network currently used in the AIR project.

5.6 Environments

The experiments are carried out using Python 3.8.5 where the library PySyft 0.2.9 is used for applying SMPC with PyTorch 1.4.0. Additionally, Opacus 0.10.1 is used for model training with differential privacy in PyTorch 1.7.0. The different PyTorch versions are necessary as the libraries are not compatible. Finally, the library SHAP 0.37.0 is used for explainability.

5.7 Other Parameters

For the SMPC experiments, the precision value of the fixed precision encoding was set to 3 which is the default. For differential privacy using Opacus, the alphas are set to the recommended: $\alpha \in \{1 + \frac{x}{10}, x = 1..100\} \cup \{12, 13, ..., 63\}$, and likewise the gradient clipping threshold is set to $V = 1$. The noise multiplier is fixed to 1.2, as this seems fitting. Reducing this value results in more precise gradients and less privacy whereas an increment lowers accuracy and provides more privacy. The value of the differential privacy constant δ is set to a value of 1×10^{-5} that is lower than the inverse size of the largest data set: $\frac{1}{|D|} = \frac{1}{2000} = 5 \times 10^{-4}$ which is recommended [50].

Chapter 6

MNIST Experiments, Results and Discussions

In this chapter, the experiments conducted on the MNIST data set are documented. First, the data will be described and then the experiments are shown afterwards.

6.1 MNIST Data Set

MNIST, provided by Yann LeCun et al. [58], is a data set consisting of images depicting handwritten digits. The data set is widely used for benchmarking machine learning algorithms, albeit it is often the case that high test scores can be achieved using rather simple models. The MNIST data set will here be used to compare the different algorithms described earlier to get a better understanding of their performance. Thereafter, the algorithms will be applied on the rehabilitation data.

The training data set of MNIST has 60,000 samples and the test set has 10,000 samples. However, due to limited computational resources only a small subset of this data is used. Of each class, we take 160 training examples and 40 test samples, giving a total of 2,000 examples where 400 (20%) of these are used as test samples. All images are grayscale with dimensions 28 by 28 pixels that range from zero to 255 in value, and each image depicts a digit from zero to nine, giving 10 different classes.

A sample of each class can be seen below in Figure 6.1.

Figure 6.1: Random sample of each class in the MNIST data set.

6.2 MNIST Experiments

The experiments that are conducted using the MNIST data set are listed here. An experiment is first conducted using a non-federated data-centralized approach to get a baseline for comparison with other algorithms. Next, federated learning with differential privacy is experimented with, followed by an experiment using federated learning with SMPC. Then, the computation times of all algorithms are documented and discussed. Experiments with different data distributions are conducted in the fifth experiment, and finally explainability is investigated.

- MNIST Experiment 1: Accuracy with Centralized Model and Data
- MNIST Experiment 2: Accuracy Using Local Differential Privacy
- MNIST Experiment 3: Accuracy using SMPC
- MNIST Experiment 4: Computation Time
- MNIST Experiment 5: Individual Client Accuracy with Local Models
- MNIST Experiment 6: Explainability

6.3 MNIST Experiment 1: Accuracy with Centralized Model and Data

6.3.1 Purpose

The purpose of this experiment is to see how well a typical non-federated data-centralized learning approach performs using the specified model architecture. The results will be used for comparison with the federated approaches later.

6.3.2 Results

The accuracy and loss during both training and testing can be seen in Figure 6.2 for different values of batch size B. The number of epochs are kept constant to allow comparison with the federated algorithms later.

6.3.3 Discussion

The centralized model seems to converge faster using lower batch sizes in both training and test metrics. When $B = 16$ the training accuracy reaches about 95% after 20 epochs, however, the test accuracy reaches less than 90% which may indicate slight overfitting. This model is selected for comparison in coming experiments, where the metrics are plotted as a function of communication rounds instead of epochs. A confusion matrix of this model can be seen in Figure A.1.

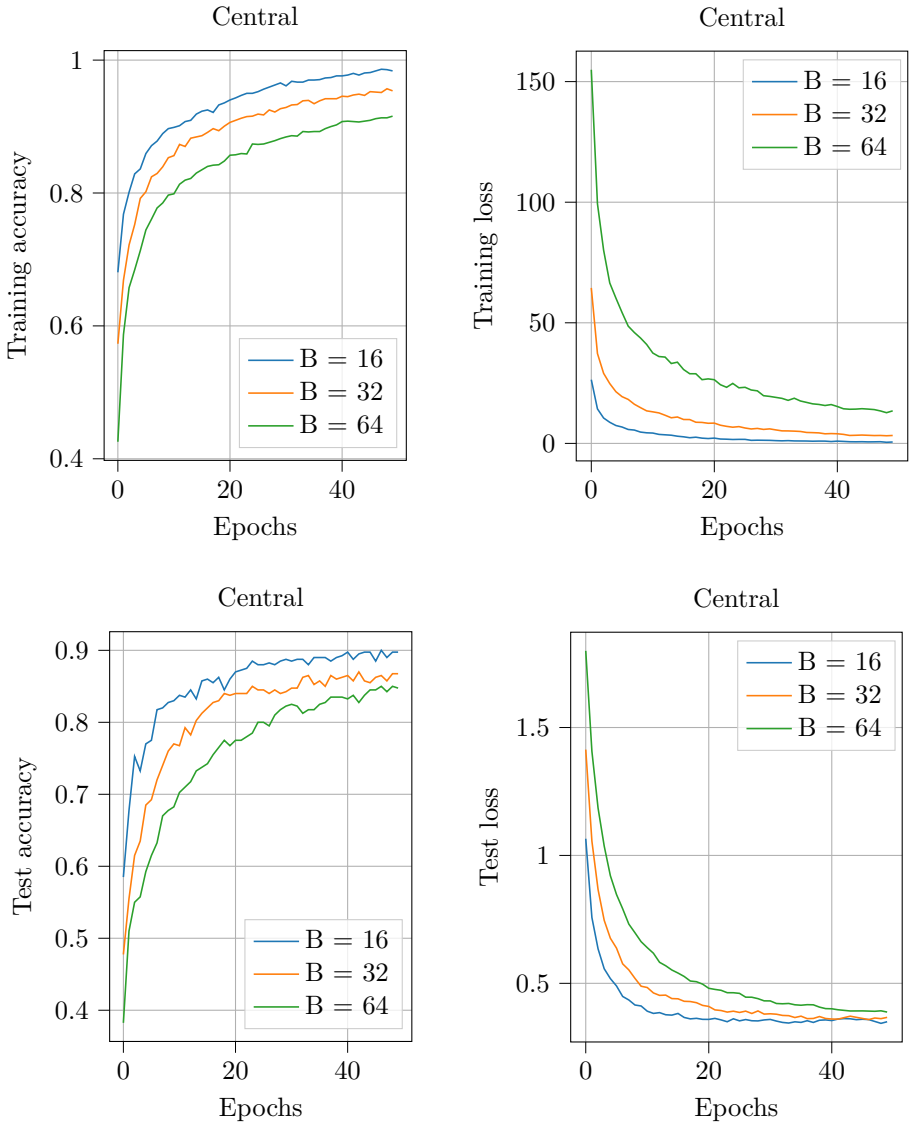


Figure 6.2: Accuracy and loss during both training and testing of a centralized model trained using centralized data. The plots were made by varying training batch size.

6.4 MNIST Experiment 2: Accuracy with Local Differential Privacy

6.4.1 Purpose

The purpose of this experiment is to see how much the locally differentially private federated algorithm affects accuracy and loss in contrast to the pure Federated Averaging algorithm in an ideal data distribution setting.

6.4.2 Results

The training accuracy and test accuracy can be seen in Figure 6.3 for various epochs, where the number of clients $K = 3$ and batch size $B = 16$. The same metrics are shown for various batch sizes in Figure 6.4. Finally, the privacy budgets ϵ are shown in Figure 6.5.

6.4.3 Discussion

We see that the differentially private methods (green and purple) perform worse than their non-differentially private counterparts (orange and red) – roughly 15 percentage points. This is the case when varying either the number of epochs or the batch size. Using a large epoch size seem to cause the Federated Averaging algorithm to stop converging both with and without differential privacy. Intuitively, this makes sense because the local models on the clients may learn too much of the local data, resulting in a lack of "agreement" when the model updates are aggregated. Finally, we see that the privacy budget of differential privacy increases for larger numbers of epochs and batch sizes. The differentially private learning algorithm adds noise per sample, which means that the privacy budget increases for every sample. Therefore, it also makes sense that increasing epochs and batch size at the same time increases the privacy budget.

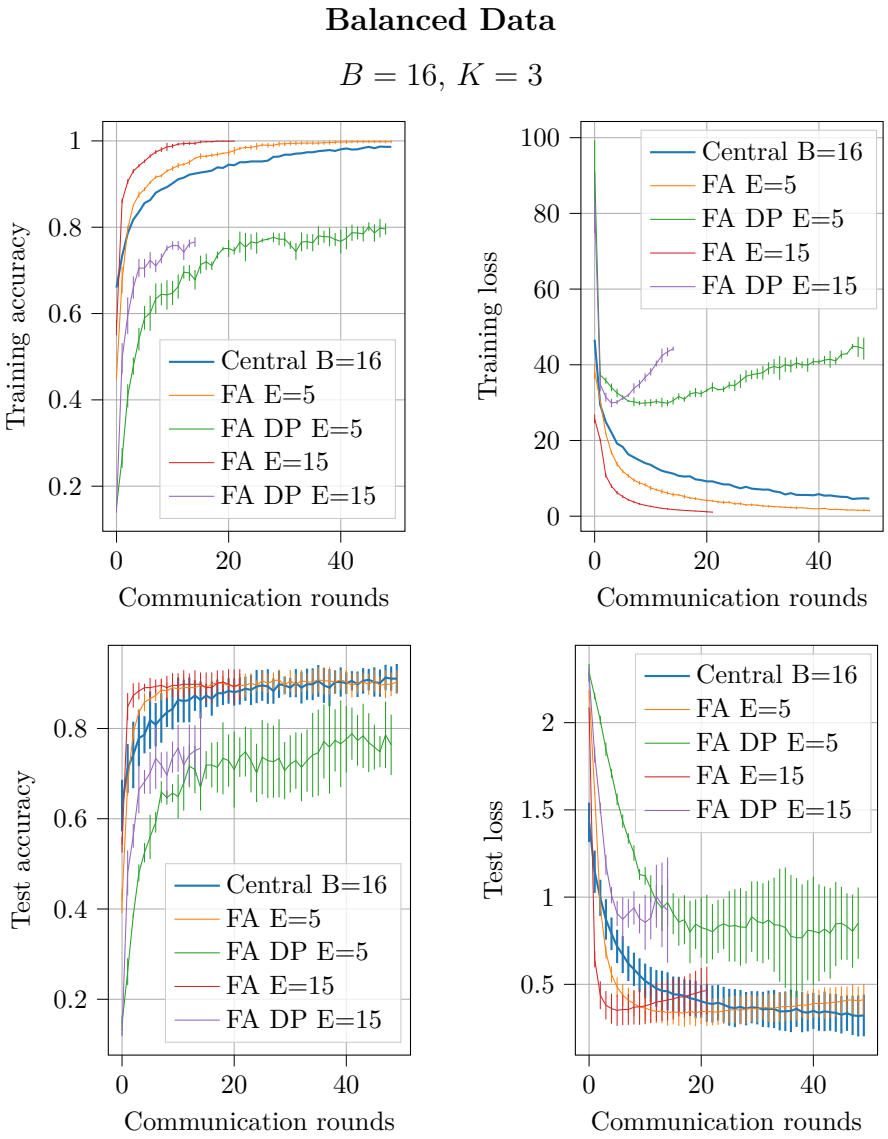


Figure 6.3: Accuracy and loss during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different numbers of epochs. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

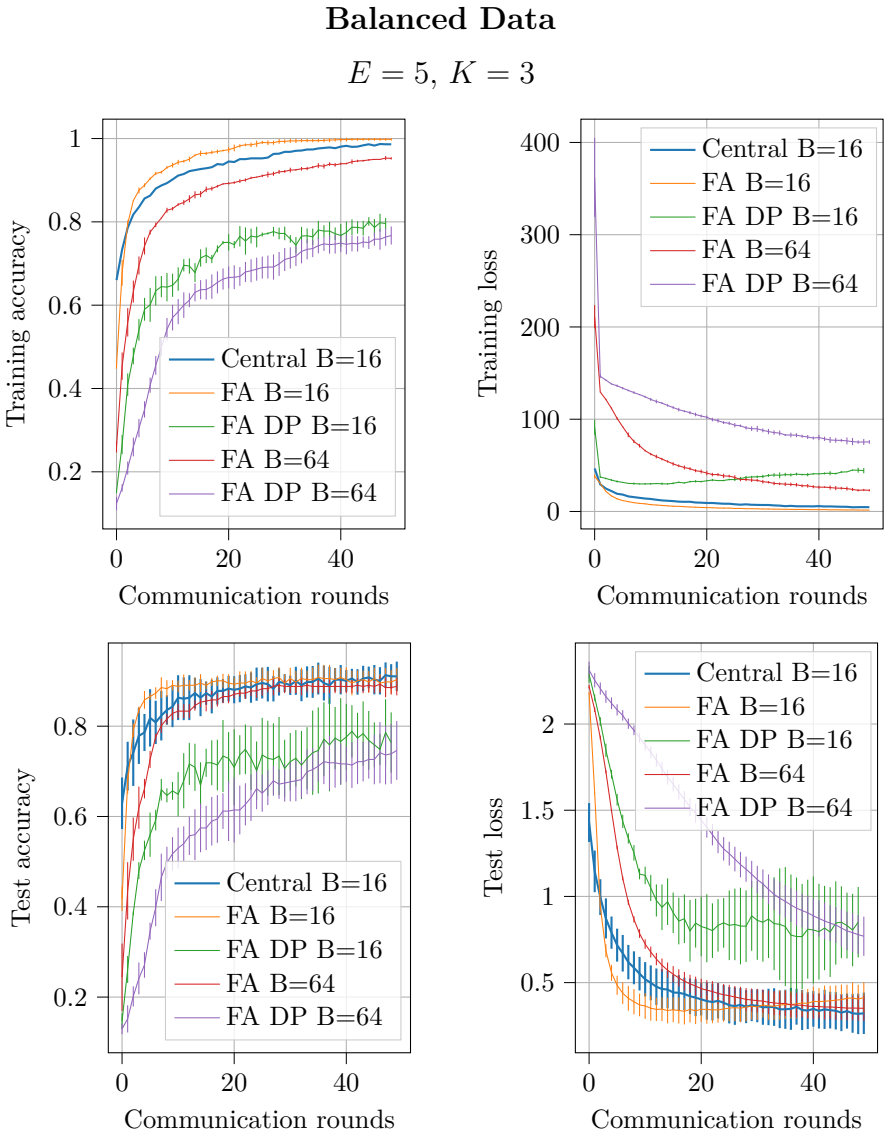


Figure 6.4: Accuracy and loss during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different batch sizes. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

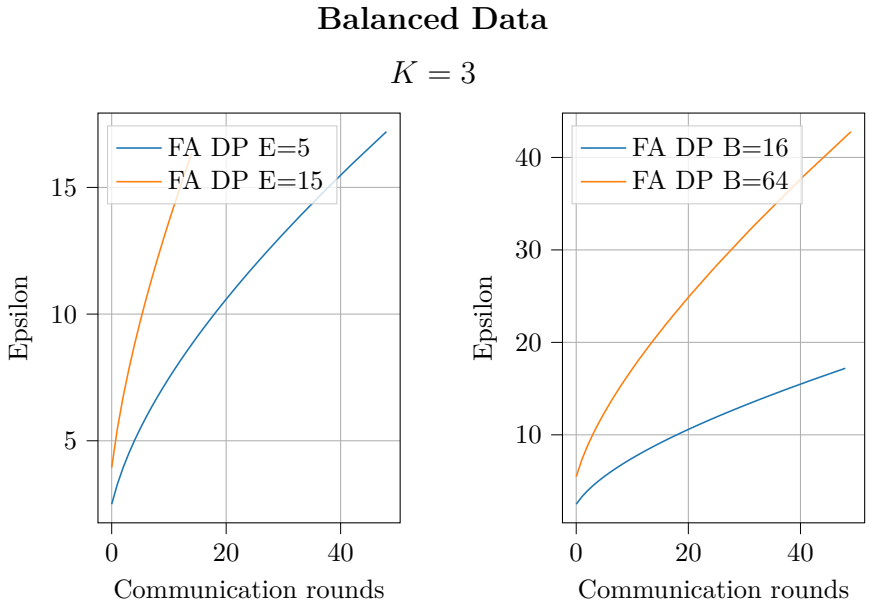


Figure 6.5: Privacy budget ϵ of a global model trained using Federated Averaging with local differential privacy for different epochs and different batch sizes. The data was balanced and not label skewed. Batch size was fixed to 16 and number of epochs was fixed to 5, respectively. Number of clients $K = 3$.

6.5 MNIST Experiment 3: Accuracy with SMPC

6.5.1 Purpose

The purpose of this experiment is to see how much Federated Averaging with SMPC affects accuracy and loss in contrast to the pure Federated Averaging algorithm.

6.5.2 Results

The training accuracy and test accuracy can be seen on Figure 6.6, where the number of clients $K = 3$ and batch size $B = 16$. The data is balanced on each client with no label skew.

6.5.3 Discussion

We see that the use of SMPC has no significant impact on the accuracy of Federated Averaging. The loss curves with and without SMPC almost lie directly on top each other. This makes sense since SMPC is only applied on the model updates of each client. If an appropriate amount of precision is used for the fixed precision encoding part of SMPC, then the model updates will be very close to the non-SMPC version – leading to very similar accuracy and loss curves as we see here. The experiment was also carried out for various batch sizes with similar results. Thus, only the results for various epoch sizes are shown.

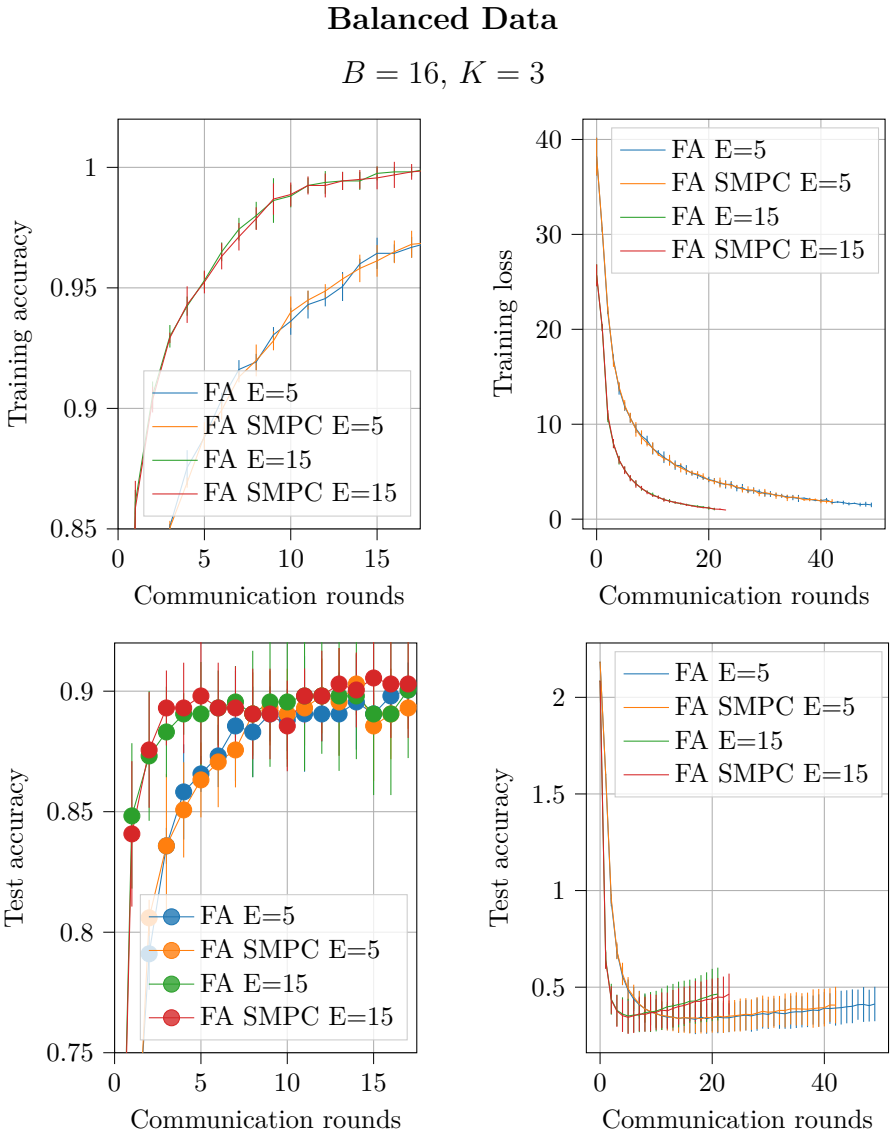


Figure 6.6: Accuracy and loss during both training and testing of a global model trained using Federated Averaging with and without SMPC for different numbers of epochs. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

6.6 MNIST Experiment 4: Computation Time

6.6.1 Purpose

The purpose of this experiment is to compare the different federated algorithms with respect to computation time.

6.6.2 Results

When varying one parameter the other parameters are set to default: $E = 5$, $B = 16$, and $K = 3$ respectively. This gives some redundancies in the results which have been removed. For this experiment the data distribution is balanced with no label skew. The computation times of Federated Averaging versus Federated Averaging with differential privacy can be seen in Table 6.1 for various epochs E, number of clients K, and batch size B. The time unit for the results are showed in seconds with standard deviations over three runs.

	E		
Algorithm	5	10	15
FA	38.43 ± 1.72	69.58 ± 0.95	101.68 ± 1.70
FA (DP)	199.27 ± 0.68	383.28 ± 4.41	569.01 ± 0.91
Increase	x5.19	x5.51	x5.60
	B		
Algorithm	16	32	64
FA	—	23.98 ± 0.29	17.51 ± 0.16
FA (DP)	—	149.40 ± 1.47	135.50 ± 0.50
Increase	x5.19	x6.23	x7.74
	K		
Algorithm	3	5	10
FA	—	37.54 ± 0.47	38.60 ± 0.57
FA (DP)	—	208.58 ± 2.02	261.38 ± 1.82
Increase	x5.19	x5.56	x6.77

Table 6.1: Computation time of Federated Averaging with and without differential privacy for various numbers of epochs E, batch size B, and clients K.

The computation times of Federated Averaging versus Federated Averaging with SMPC can be seen in Table 6.2. Since different environments are used for SMPC and differential privacy, the pure Federated Averaging algorithm is run for both environments.

	E		
Algorithm	5	10	15
FA	586.43 ± 1.30	1145.73 ± 30.54	1687.78 ± 5.27
FA (SMPC)	587.32 ± 2.90	1179.61 ± 3.20	1702.23 ± 36.90
Increase	x1.00	x1.03	x1.01

	B		
Algorithm	16	32	64
FA	—	303.95 ± 1.17	162.95 ± 1.06
FA (SMPC)	—	305.48 ± 2.87	166.17 ± 0.68
Increase	x1.00	x1.01	x1.02

	K		
Algorithm	3	5	10
FA	—	580.84 ± 9.31	582.01 ± 2.50
FA (SMPC)	—	593.76 ± 4.68	644.89 ± 38.29
Increase	x1.00	x1.02	x1.11

Table 6.2: Computation time of Federated Averaging with and without SMPC for various numbers of epochs E, batch size B, and clients K.

Algorithm	$B = 16$		
Central	20.468 ± 1.13		

	E		
Algorithm	5	10	15
FA (PT1.4)	x28.66	x55.99	x82.48
FA (PT1.7)	x2.31	x4.19	x6.12

	B		
Algorithm	16	32	64
FA (PT1.4)	x28.66	x14.85	x7.96
FA (PT1.7)	x2.31	x1.44	x1.05

	K		
Algorithm	3	5	10
FA (PT1.4)	x28.66	x28.39	x28.44
FA (PT1.7)	x2.31	x2.26	x2.32

Table 6.3: Increase in computation time of Federated Averaging for various numbers of epochs E, batch size B, and clients K compared to a centrally trained model where $B = 16$.

6.6.3 Discussion

The experiment results in Table 6.1 show that the differentially private Federated Averaging algorithm requires more than five times the computation time of the pure Federated Averaging algorithm. This ratio grows when increasing either of the three parameters E, B, and K.

In the case of SMPC in Table 6.2, the increase in computation is roughly the same as the pure Federated Averaging algorithm which indicates that the SMPC algorithm is very performant when varying the different parameters. However, it should be noted that the values shown in the tables above do not account for the communication delays of Federated Averaging. Since the computation was carried out on a single compute node where clients were processed sequentially, and the data was kept constant even for increasing numbers of clients, these timings for various values of K may be misleading. In a real federated system, increasing the number of clients (with similar round times) should not have an impact on the pure Federated Averaging algorithm since it operates in a lockstep, or synchronized, fashion where the next round is only started when all clients have responded to the server with their model update.

Finally, in Table 6.3 the computation times of the federated averaging algorithms are compared to the central case where $B = 16$. We see that the use of Federated Averaging with PyTorch 1.4 (including PySyft with VirtualWorkers for SMPC) increases computation time much greater than the case where pure PyTorch 1.7 is used (used for differential privacy).

6.7 MNIST Experiment 5: Individual Client Accuracy with Local Models

6.7.1 Purpose

In this experiment the clients each have a locally trained neural network that is not aggregated at any point as in Federated Averaging. The purpose of this experiment is to understand how a global model trained using Federated Averaging performs in relation to these locally trained models under various data distributions. The centrally trained model is again used as the overall baseline.

6.7.2 Results & Discussions

The data can be balanced or unbalanced such that some clients have much more data than others. Additionally, the label distribution can be even or skewed such that some clients have more of certain labels and less of others. This gives four combinations of data distributions that are experimented with. Each line in the plots of this section corresponds to a single client, and different colors represent different parameters or learning approaches.

Balanced Data

In this case the data is balanced across clients and the label distribution is even. The accuracy and loss during both training and test are seen in Figure 6.7 for various epochs. The same metrics can be seen for various batch sizes in Figure 6.8.

In Figure 6.7, when $E = 15$ the global model trained using Federated Averaging (green) stops being able to learn after some communication rounds. However, for all clients with a global model the training and test accuracy reaches the levels of the central case in slightly fewer communication rounds. In Figure 6.8 we see that using a low batch size of $B = 16$ instead of $B = 64$ reduces the amount of communication rounds needed to reach about the same test accuracy.

With respect to the training metrics, the local models (red and purple) reach higher accuracies and lower losses than their counterparts trained using Federated Averaging while requiring fewer communication rounds. Nevertheless, the federated models reach more consistent high test accuracies that sometimes even surpass the central baseline.

Unbalanced Data

In this case the data is unbalanced across clients and the label distribution is even. The accuracy and loss during both training and testing are seen in Figure 6.9 for various epochs. The same metrics can be seen for various batch sizes in Figure 6.10.

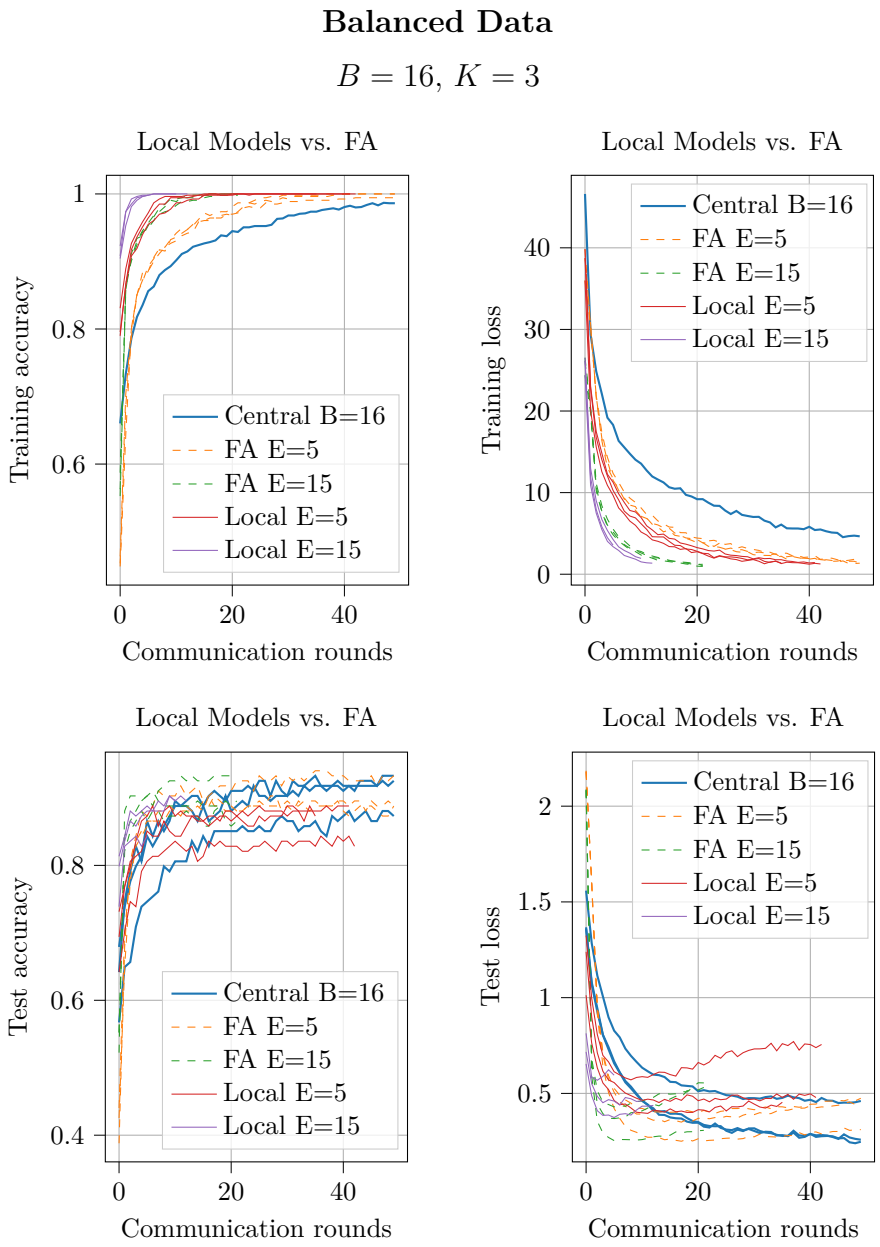


Figure 6.7: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

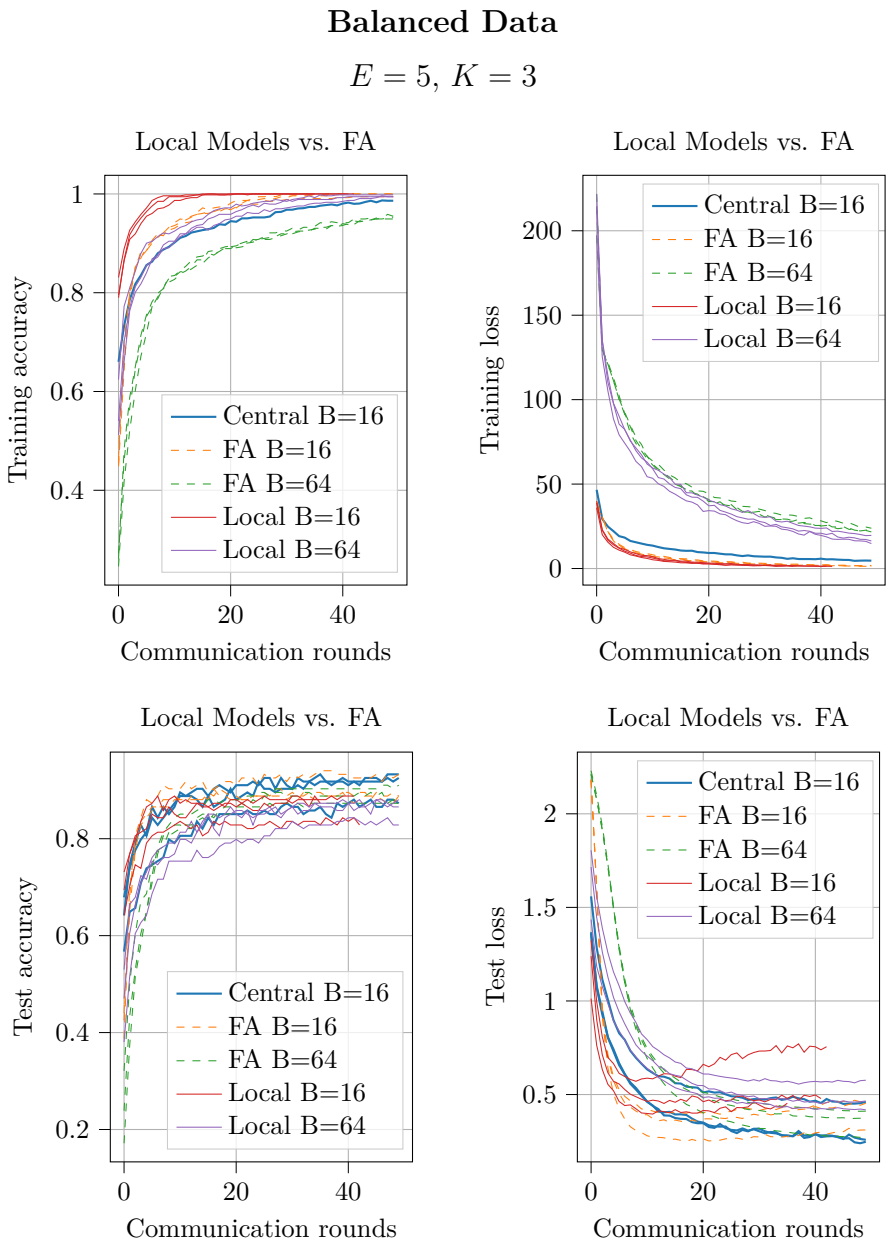


Figure 6.8: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. The data was balanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

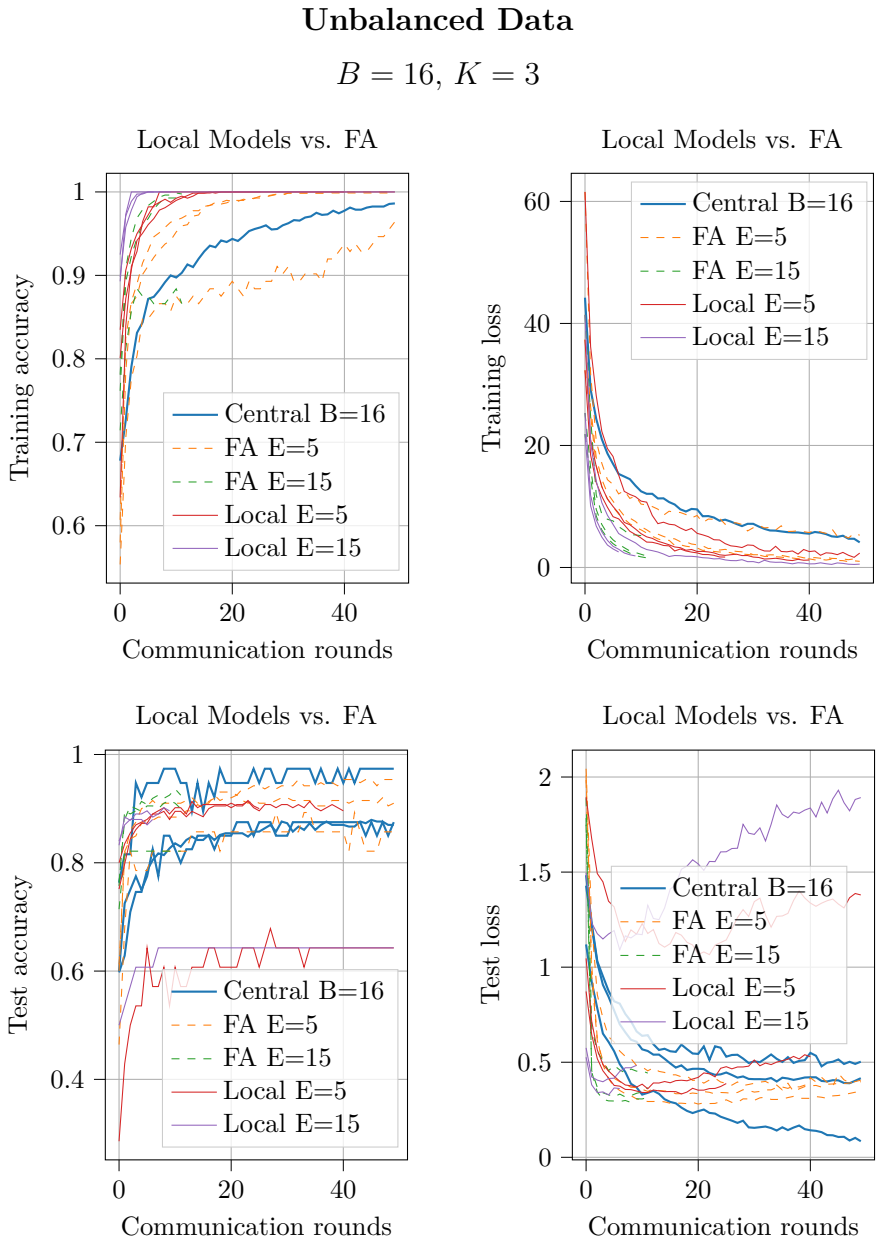


Figure 6.9: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. The data was unbalanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

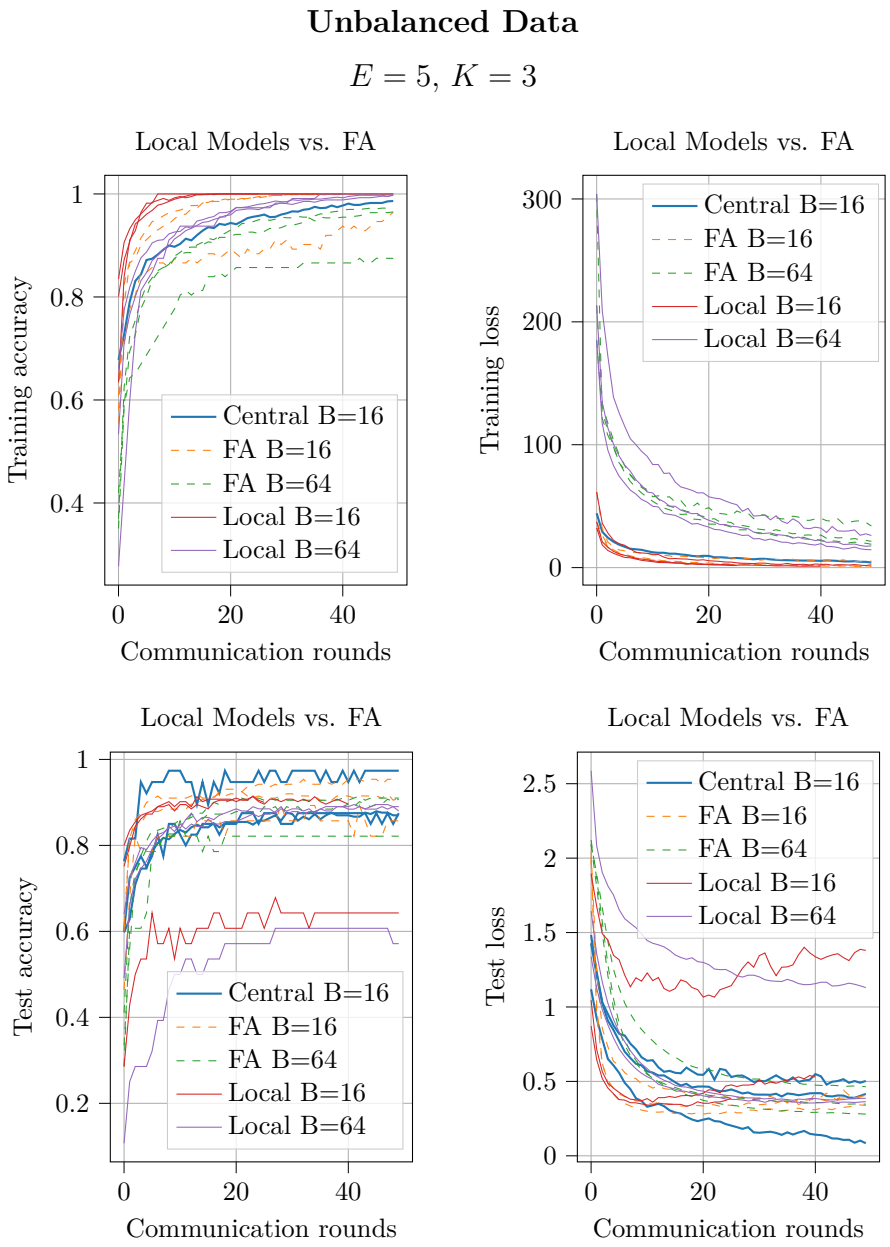


Figure 6.10: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. The data was unbalanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

In test accuracy, the local models do not perform equally well on all clients if the clients have unbalanced amounts of data. Looking in the experiment logs, the lines with low accuracy in Figure 6.9 and Figure 6.10 indeed correspond to the clients with little data. These clients perform well with regards to training accuracy but fails to generalize as seen in the test accuracies. On the other hand, we see that Federated Averaging performs better than the local models in test when varying either epochs or batch size. This indicates that models trained on clients with more data can learn patterns that are beneficial for clients with less data when the data is without label distribution skew.

Balanced Data with Label Distribution Skew

The accuracy and loss during both training and testing are seen in Figure 6.11 for various epochs. The same metrics can be seen for various batch sizes in Figure 6.12.

Comparing with the previous case, we see that models trained locally on each client seem more reliant on having more data rather than having an even label distribution across clients. Despite good performance for unbalanced data, the training and test accuracy drop for some of the federated models in this case of label distribution skew. Looking in the training logs, the clients corresponding to poorer performance seem to have greater label distribution skew. The sample count for many labels are in the range from 0 to 10 for these clients. For other clients the counts are often 0 or greater than 20.

Unbalanced Data with Label Distribution Skew

The accuracy and loss during both training and testing are seen in Figure 6.13 for various epochs. The same metrics can be seen for various batch sizes in Figure 6.14.

With respect to training accuracy, we see that the models trained locally perform better than the global models trained using Federated Averaging. The local models reach higher accuracies in fewer communication rounds. Nevertheless, the test performances of a few local models are 10 to 15 percentage points lower with increasing test loss after a few rounds.

The model aggregation of Federated Averaging seems to stabilize the learning process which removes the "outlier" client models. The federated models where $E = 5$ and $B = 16$ (yellow) performs well for all clients. Surprisingly, the model performance resembles that of the balanced case, albeit, with about 5 percentage points lower test accuracies when varying either epochs or batch size. Furthermore, the models achieve comparable performance to the central baseline model.

A summary of performance in all four data distribution cases can be found in Table 6.4.

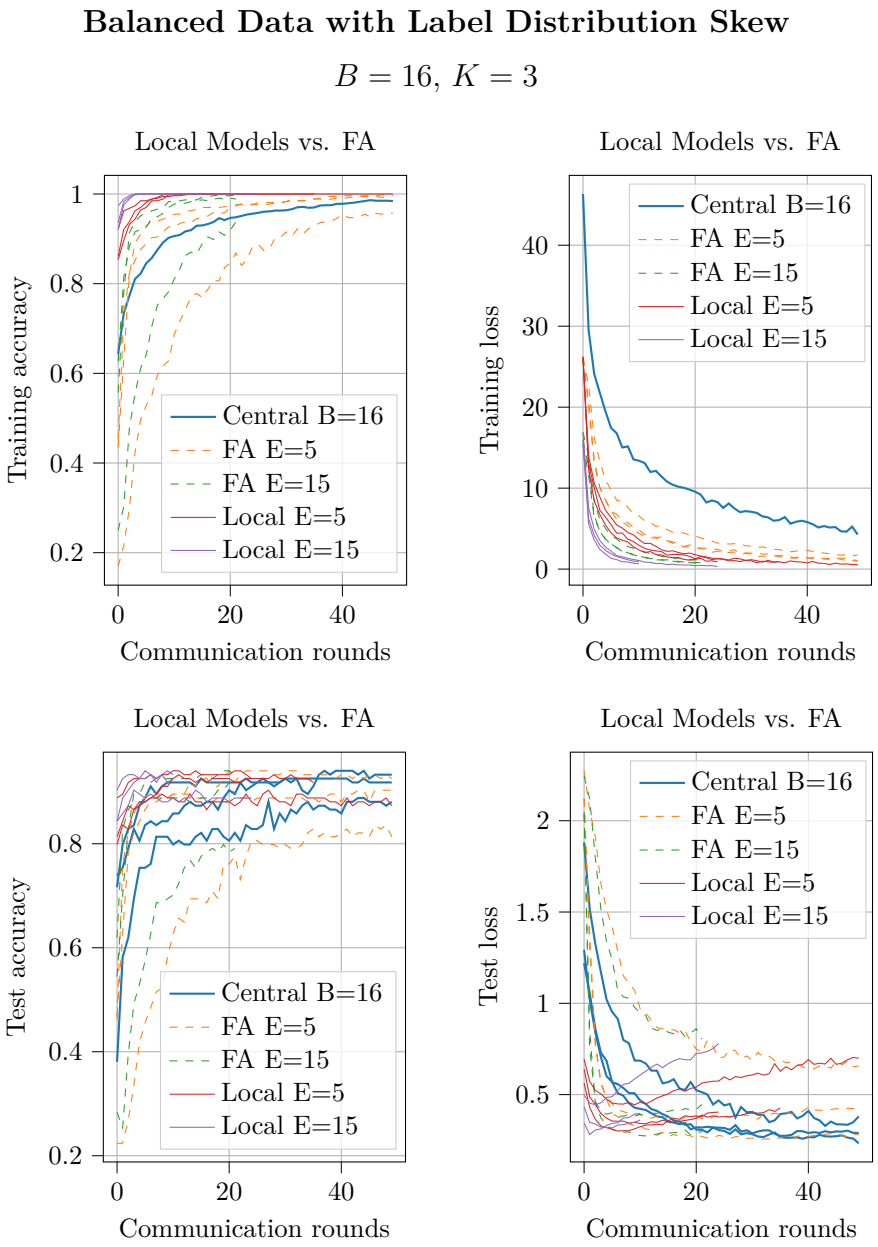


Figure 6.11: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. The data was balanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

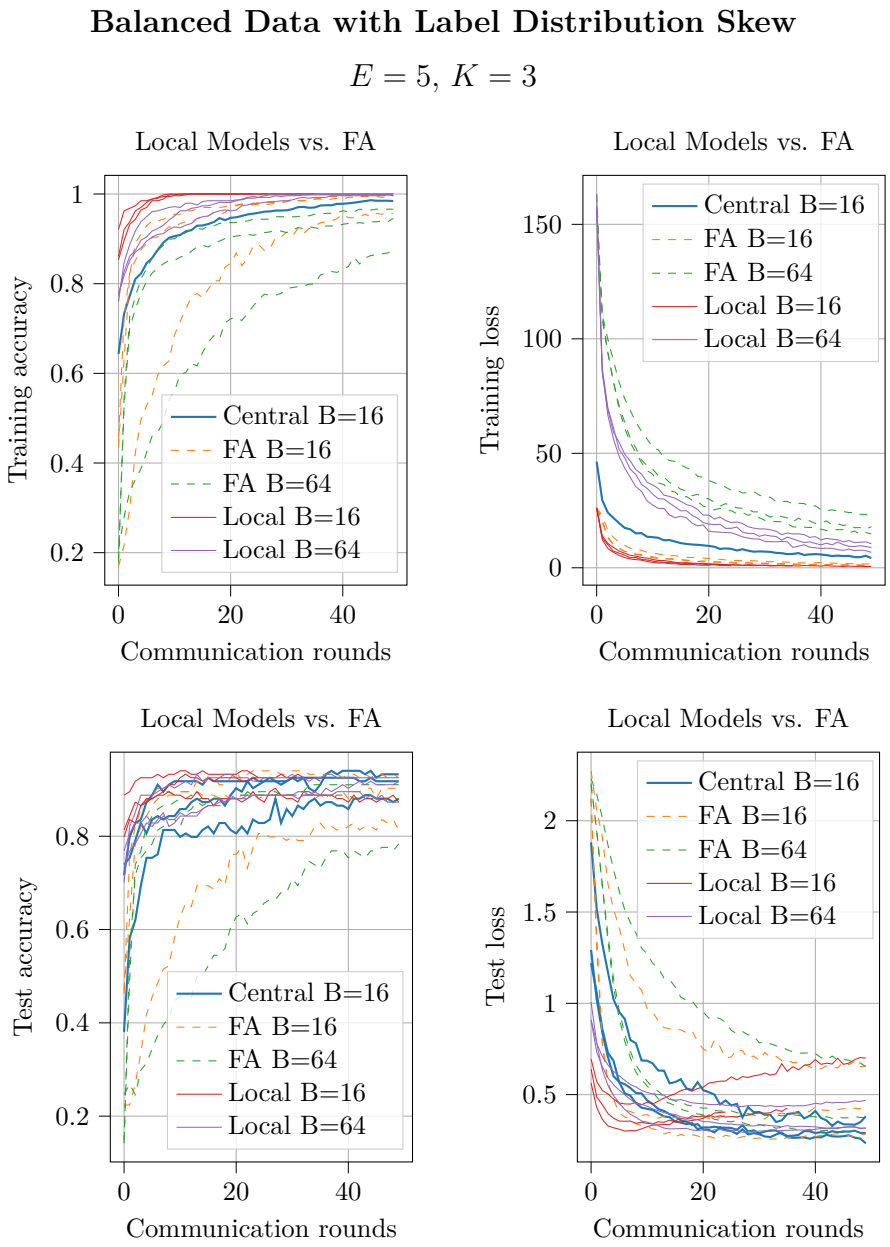


Figure 6.12: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. The data was balanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

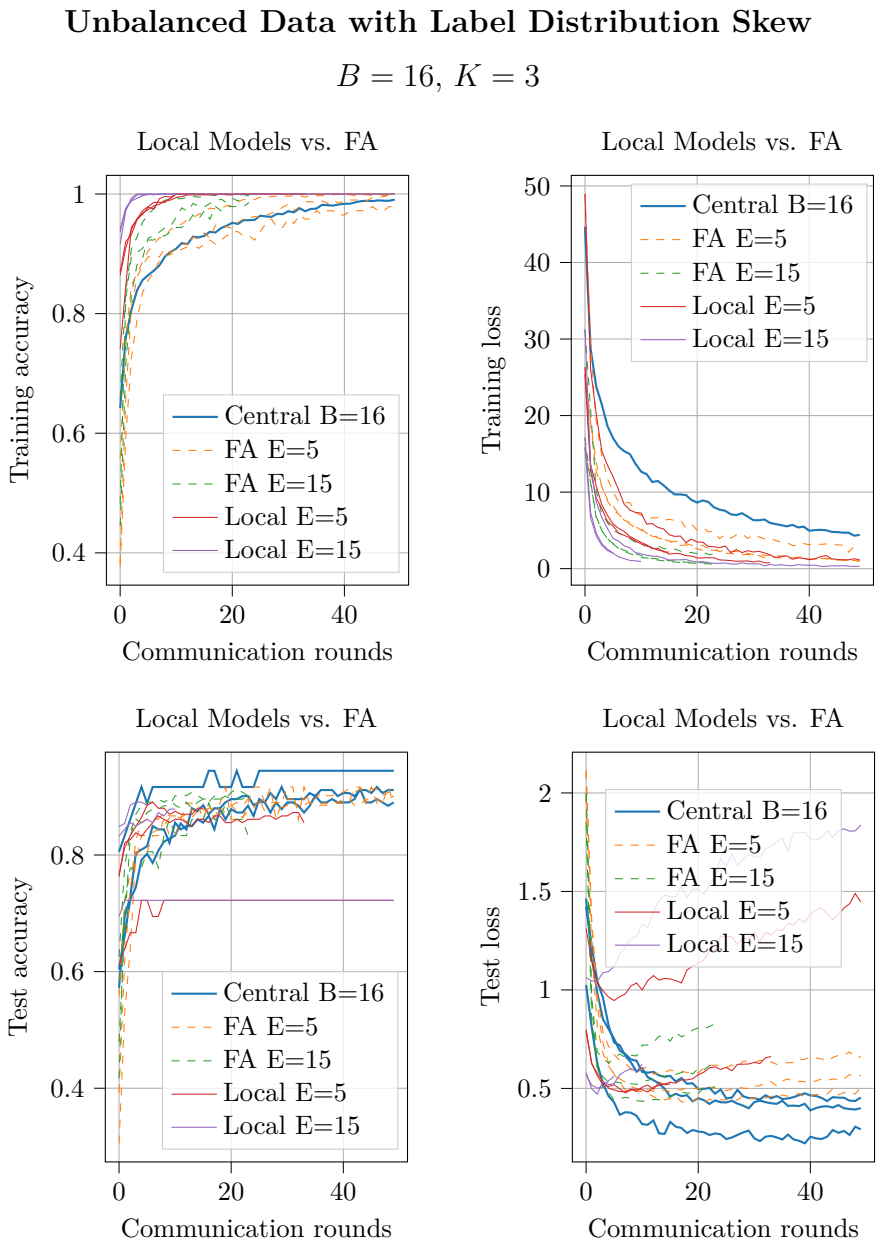


Figure 6.13: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. The data was unbalanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

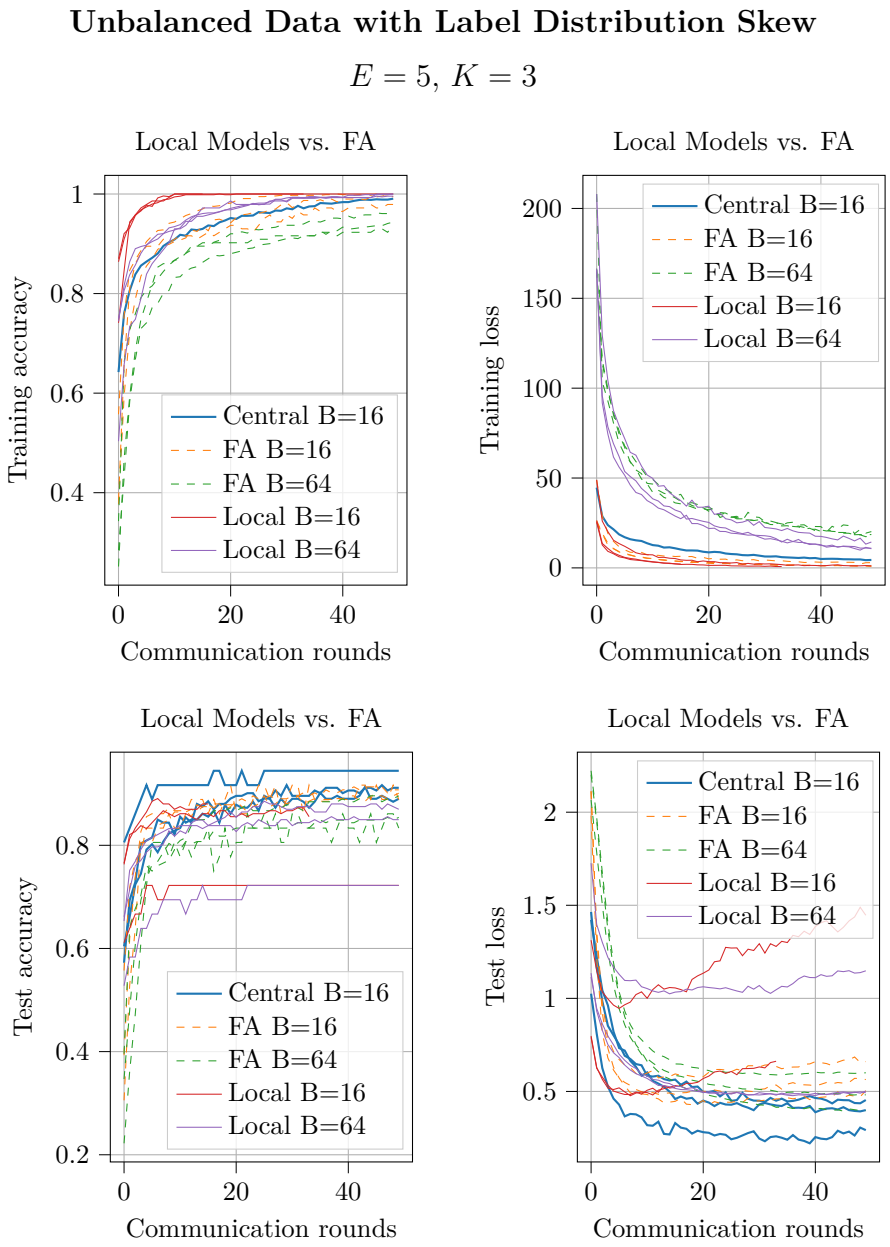


Figure 6.14: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. The data was unbalanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

MNIST Data Set	
Data Distribution	Performance Summary
Balanced	The federated models consistently reach high test accuracy scores above locally trained models. The scores are comparable to the non-federated model.
Unbalanced	The federated models are again able to reach high accuracy scores comparable to the non-federated model, whereas the locally trained models struggle when only a low amount of data is available on a client.
Balanced with label distribution skew	Some of the federated models achieve lower accuracy than the worst-performing client of the baseline. In this case the models trained locally on each client perform better than both the central model and the federated models which makes sense because the local models can personalize to the specific client it is trained on.
Unbalanced with label distribution skew	Surprisingly, the federated models perform similarly to the non-federated model despite training on data with label distribution skew. The locally trained models are difficult to train on clients with low amounts of data.

Table 6.4: Summary of results regarding Federated Averaging on different data distributions of the benchmarking data set, MNIST.

6.8 MNIST Experiment 6: Explainability

6.8.1 Purpose

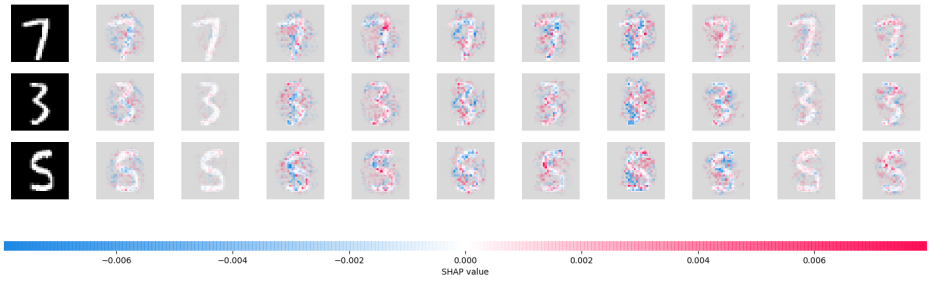
The purpose of this experiment is to see how significantly the explainability changes when changing from a data-centralized learning approach to a federated approach using Federated Averaging.

6.8.2 Results

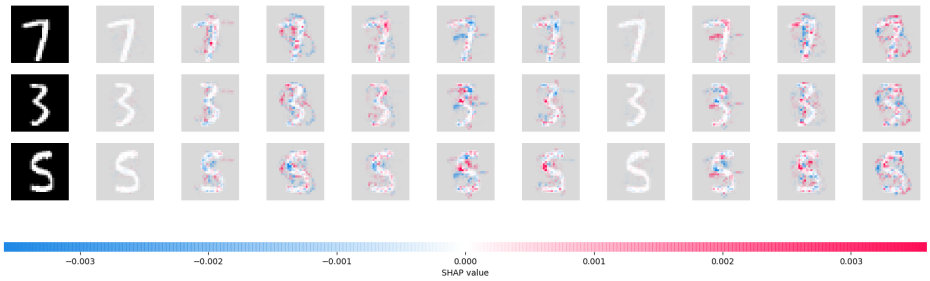
A SHAP image plot of a traditional model trained with centralized data is shown in Figure 6.15 along with two SHAP image plots from models trained with Federated Averaging with and without differential privacy.

6.8.3 Discussion

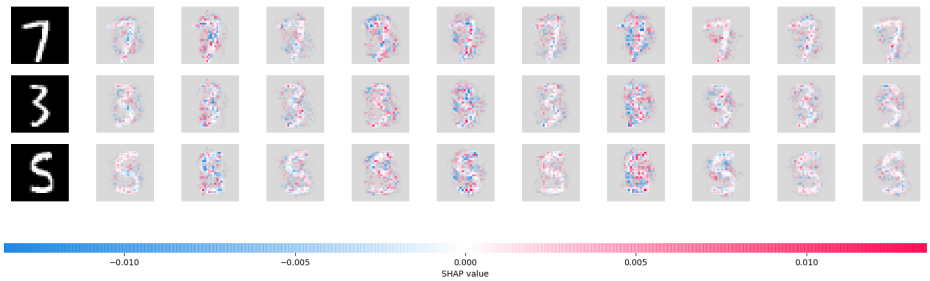
We see that the images are rather difficult to read. This is likely due to the use of a fully connected feedforward neural network (MLP), since the flattened input vector does not provide spatial information. However, we do see that Federated Averaging with differential privacy and the central model seem to give more noisy SHAP values than pure Federated Averaging.



(a) Model trained using centralized data.



(b) Federated Averaging $E = 5$, $B = 16$, and $K = 3$.



(c) Federated Averaging with differential privacy $E = 5$, $B = 16$, and $K = 3$.

Figure 6.15: SHAP image plots using models trained with (a) centralized data, (b) Federated Averaging and (c) Federated Averaging with differential privacy on MNIST data.

Chapter 7

Real Data Experiments, Results and Discussions

In this chapter, the experiments conducted on the real rehabilitation data are described. First the data set is explained and then the different experiments are progressively described.

7.1 Rehabilitation Program Completion Data Set

The real rehabilitation data set used for this thesis is provided by Aalborg Municipality and DigiRehab. DigiRehab delivers an intelligent digital exercise tool for social workers that allows tracking of citizens' potential for rehabilitation. The tool is also able to calculate training programs that are specific for the individual citizens by using data from standardized screenings of the citizens. When DigiRehab performs screenings of citizens, data about the citizen is recorded in a database. This allows the care givers to track the rehabilitation progress of individual citizens. In this manner, physiotherapy-based rehabilitation data has been collected of many citizens over the duration of many years. We would like to be able to identify, or predict, the citizens who are most likely to complete a physiotherapy-based rehabilitation program successfully as early as possible. Therefore, we only make use of the data available after the first screening from DigiRehab after their first visitation. This is also currently done in the AIR project. The provided data set is tabular and consists of 27 feature columns and 1041 rows. Furthermore, the data set has been anonymized in advance by DigiRehab such that no citizen can be directly identified. The features of the data set are:

1. **Gender**. A binary value indicating the gender of the citizen.
2. **BirthYear**. The last two digits of the citizen's birth year.
3. **MeanEvaluation**. The mean evaluation score of all exercises that a citizen has performed during screening sessions. This score varies from zero to six.
4. **NumberFalls**. The registered number of times the given citizen has fallen.
5. **NumberAts**. The amount of assistive devices that a citizen has acquired.

6. **Needs**. "Need for help" score given by answering 10 different questions during a screening session. This score indicates how much the citizen is reliant on help from others in daily activities, and ranges from zero to 100. Thus, a higher score means more reliance on others, and a lower score indicates more self-reliance.

7. **Physics**. Physical strength score measured during screening by letting the citizen perform ten test exercises. Similar to the "need for help score" this score also ranges from zero to 100.

8. **NumberExercises**. The number of exercises a citizen has received at the first screening session with DigiRehab. Citizens can receive from zero to 10 exercises per screening session.

9. **1Ex to 9Ex (9 columns)**. Feature embedding values for the first ten exercises a citizen receives.

10. **1Ats to 10Ats (10 columns)**. Feature embedding values for the first ten assistive devices a citizen has acquired.

11. **Complete**. A binary value indicating whether or not the citizen has completed the rehabilitation program. Completion is here defined as following the rehabilitation program for at least 8 weeks out of the total 12 week long training program, and additionally the citizen must participate in at least 10 training sessions during those eight weeks. More specifically, the value is set to 1 if the particular citizen has completed the rehabilitation program. Otherwise, the value is zero. This is the value to be learned and predicted by the models.

The exercises and the assistive devices were originally stored in the data set as ordered collections of string IDs. One-hot encoding these IDs resulted in high-dimensional vectors. The feature embeddings of the data set (exercises and assistive devices) are lower-dimensional representations of these vectors such that similar inputs vectors are placed closer in the lower-dimensional space. This can make it easier to get good predictive performance when doing machine learning [59].

The rehabilitation data is rather well-balanced with roughly half ($494/1041 \approx 47.5\%$) of the citizens being able to complete their rehabilitation program. A small section of the data set can be seen in Table 7.1. It is here assumed that each citizen has only one row of data that is sensitive to them because (ϵ, δ)-differential privacy is designed to protect privacy between adjacent databases that only differ in a single row. It is possible to define differential privacy over databases that can differ in multiple rows (group privacy), however, no relevant literature could be found for this kind of user-level privacy in a cross-silo setting.

Gender	BirthYear	Mean Evaluation	Number Falls	Number Ats	Needs	Physics
1	33	0.0	0	7	39	49
1	50	0.0	2	24	19	20
0	30	4.0	0	9	89	29
0	30	4.0	0	14	10	29

Number Exercises	1Ex	2Ex	...	9Ats	10Ats	Complete
7	-0.0211	-0.3551	...	-0.0017	0.0951	0
5	-0.0161	-0.0679	...	0.0019	-0.1654	0
5	0.2318	-0.3551	...	0.1755	0.0951	1
6	0.2318	0.2093	...	0.1755	-0.4145	1

Table 7.1: A section of the rehabilitation data set.

7.2 Real Data Experiments

The same experiments as for the MNIST data set are performed here on the real rehabilitation data set. Though, with the exception of the experiment with SMPC due to the results found in the MNIST experiment. Thus, the experiments that are conducted using the rehabilitation data are:

- Real Data Experiment 1: Accuracy with Centralized Model and Data
- Real Data Experiment 2: Accuracy Using Local Differential Privacy
- Real Data Experiment 3: Computation Time
- Real Data Experiment 4: Individual Client Accuracy with Local Models
- Real Data Experiment 5: Explainability

7.3 Real Data Experiment 1: Accuracy with Centralized Model and Data

7.3.1 Purpose

The purpose of this experiment is to see how well a typical non-federated data-centralized learning approach performs using the specified model architecture. The results will be used for comparison with the federated approaches later.

7.3.2 Results

The accuracy and loss during both training and testing can be seen on Figure 7.1 and AUC values are shown in Figure 7.2. The number of epochs are kept constant to allow comparison with the federated algorithms later.

A confusion matrix of this model can be seen in Figure A.2. Precision and recall values have also been measured for comparison with the federated models. However, for brevity, these are also shown in the appendix in Figure B.1.

7.3.3 Discussion

The centralized model with lower batch size $B = 16$ reaches better scores in all metrics including precision and recall during both training and test. This model is selected for comparison in coming experiments.

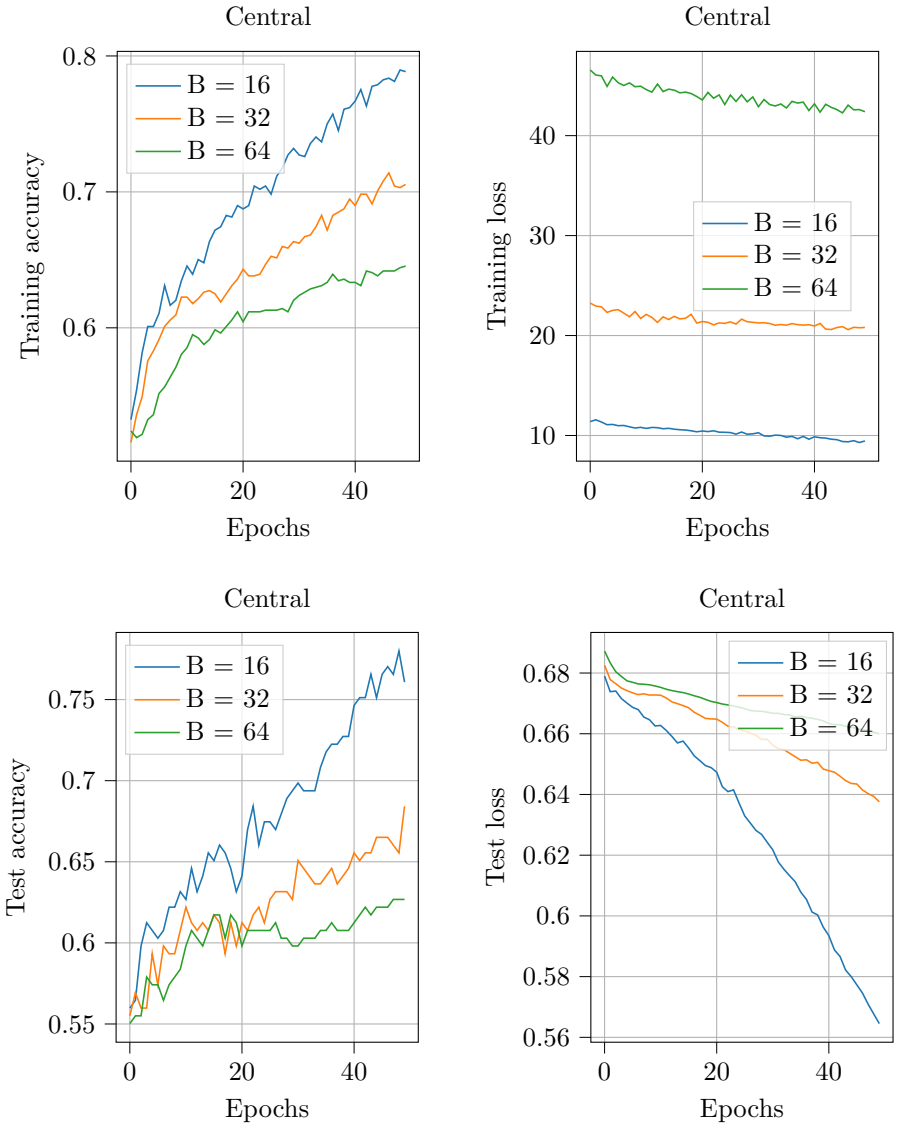


Figure 7.1: Accuracy and loss during both training and testing of a centralized model with centralized rehabilitation data. The plots were made by varying training batch size.

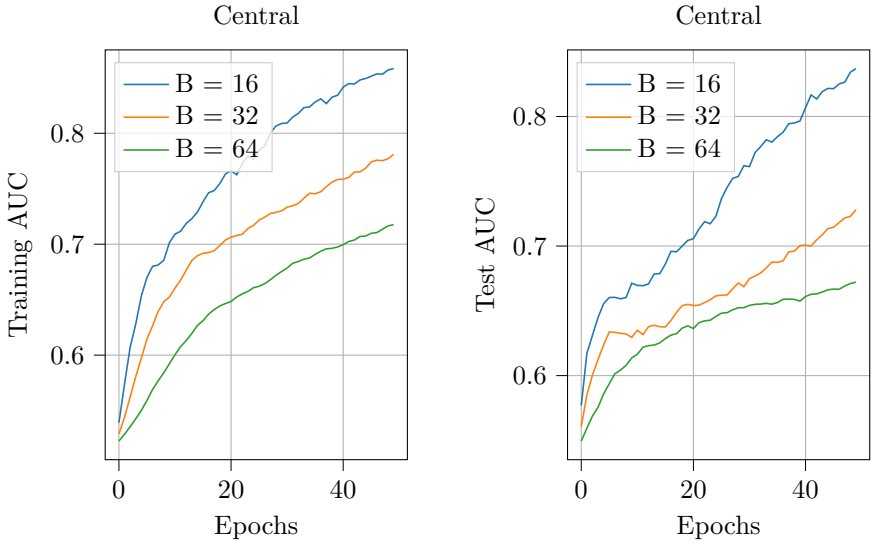


Figure 7.2: AUC during both training and testing of a centralized model with centralized rehabilitation data. The plots were made by varying training batch size.

7.4 Real Data Experiment 2: Accuracy with Local Differential Privacy

7.4.1 Purpose

The purpose of this experiment is to see how well the locally differentially private federated algorithm performs in contrast to the pure Federated Averaging algorithm in an ideal data distribution setting.

7.4.2 Results

First the results for varying epochs E are shown, where the number of clients $K = 3$ and batch size $B = 16$. The data in this case is balanced on each client with no label skew. Accuracy and loss during both training and testing can be seen in Figure 7.3 and Figure 7.4 for various epochs and batch sizes respectively. Likewise, AUC scores can be seen in Figure 7.5. The ϵ privacy budgets are plotted in Figure 7.6.

Finally, the precision and recall values can be seen in Figure B.2 and Figure B.3 in the appendix.

7.4.3 Discussion

In Figure 7.3 we see that when the number of epochs $E = 5$, the differentially private method (green) achieves roughly 12 percentage points less in accuracy than the non-differentially private counterpart (yellow) in both training and test. However, when $E = 15$ the differentially private method (purple) is within about 5 percentage points in accuracy of the pure federated algorithm (red). Additionally, we see that the pure federated algorithms reaches the same test accuracy as the data-centralized model (blue) or slightly higher. In Figure 7.4 the results show that increasing batch size on the real data set to $B = 64$ reduces the accuracy with and without differential privacy. In general the use of differential privacy makes it harder for the model to learn, since noise is added through the differentially private SGD algorithm. Due to the added noise, we also see higher variance for these algorithms.

In Figure 7.5 the test AUC for Federated Averaging with differential privacy $E = 5$ (green) reaches about 71%. In all other cases the test AUC reaches about 84%. When raising the batch size to $B = 64$ the test AUC drops to 60% for both the private and non-private algorithms. Using a lower batch size is seemingly better in this case.

The privacy budget of differential privacy grows for increasing epochs and batch sizes as seen in MNIST experiment 2. Because the differentially private algorithm uses the moments accountant of Abadi et al. the privacy budget can be reduced (improved) by increasing data set size [32, 33].

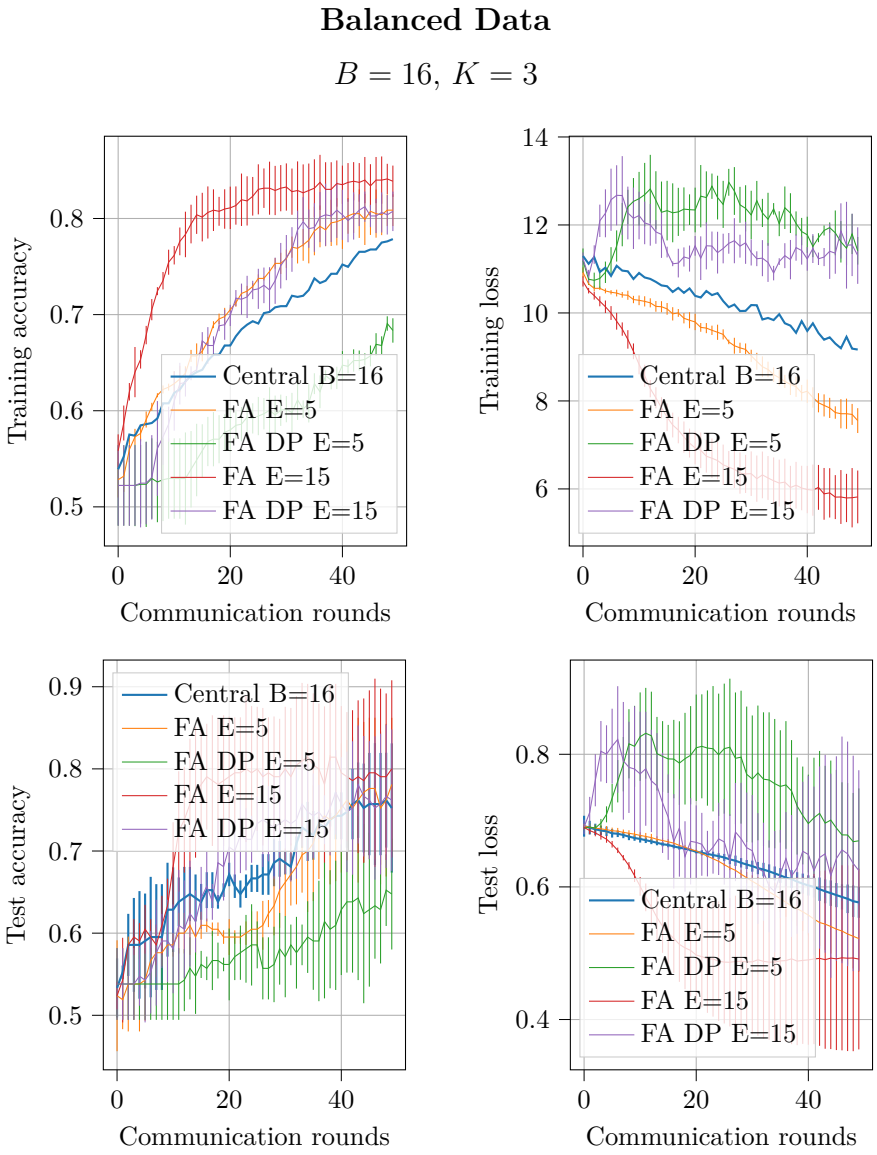


Figure 7.3: Accuracy and loss during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different numbers of epochs. Training was done using the rehabilitation data. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

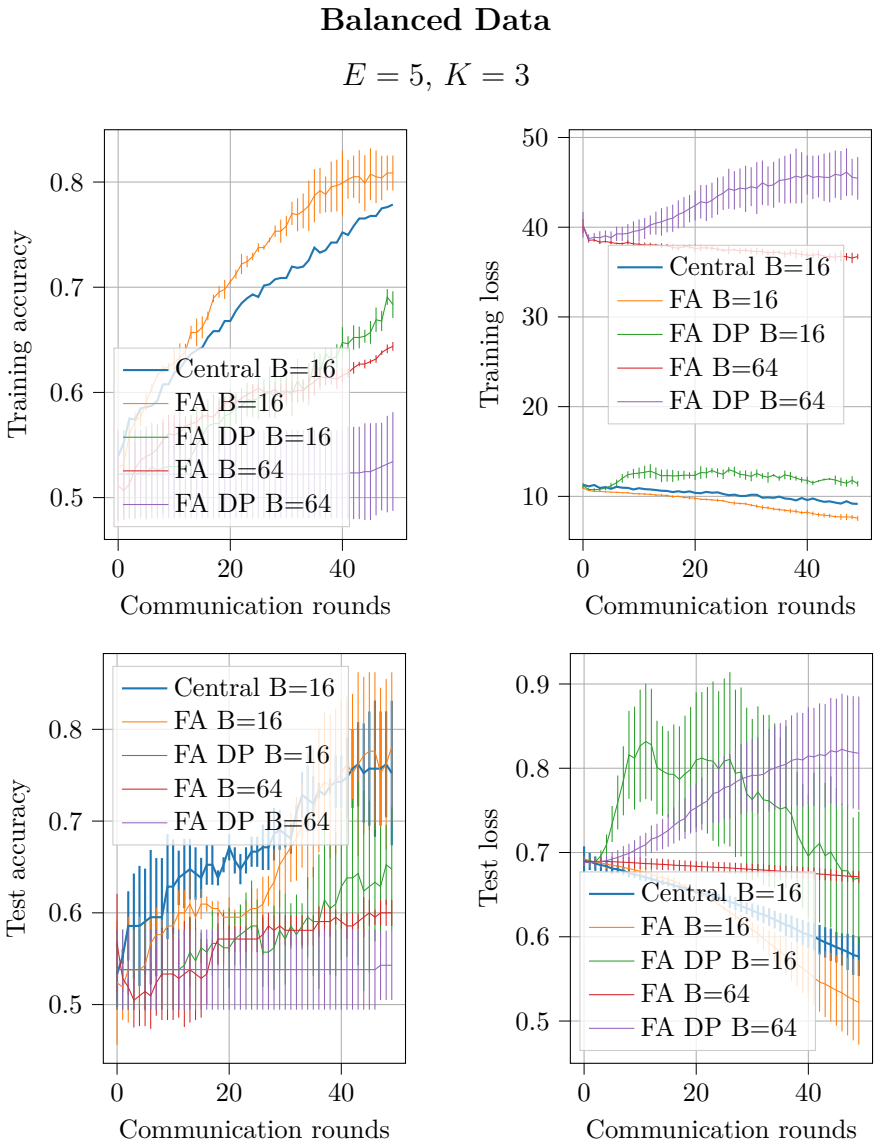


Figure 7.4: Accuracy and loss during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different batch sizes. Training was done using the rehabilitation data. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

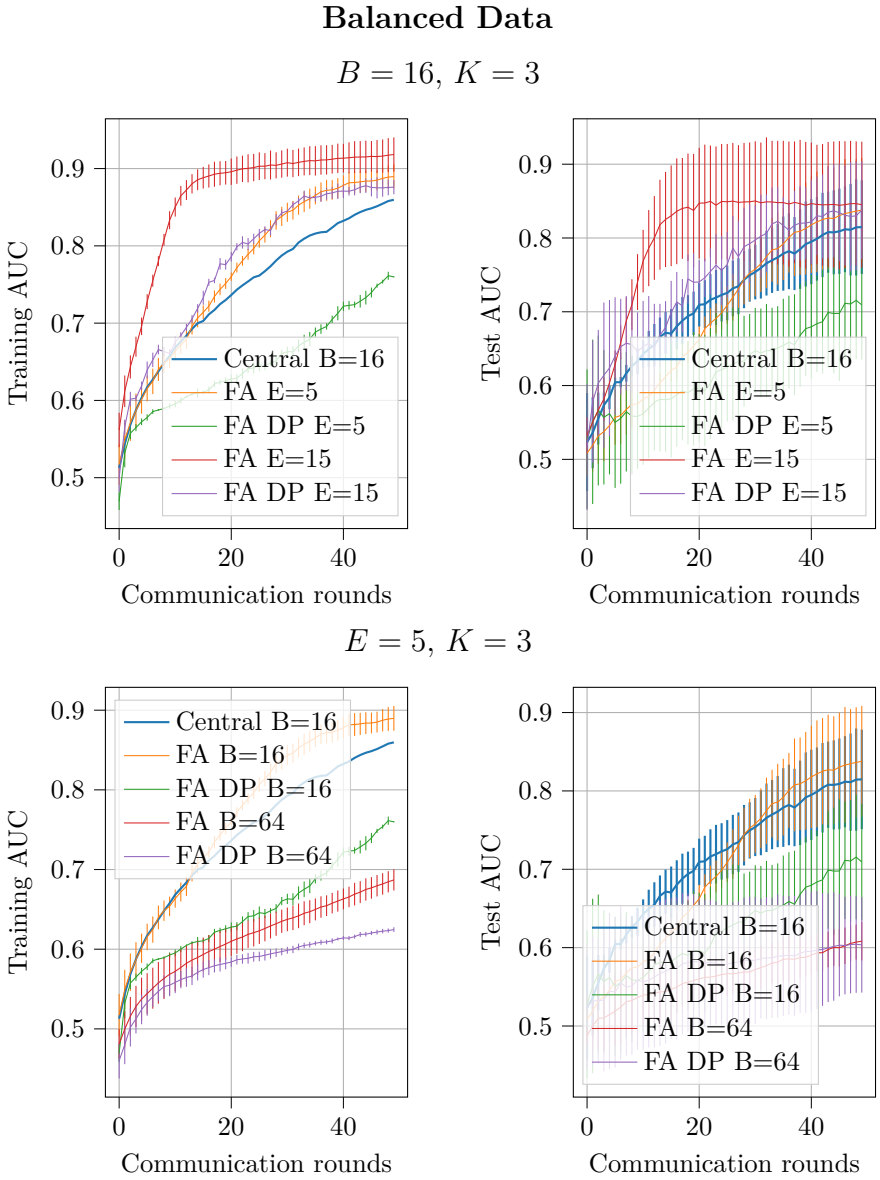


Figure 7.5: AUC during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different numbers of epochs and batch sizes. Training was done using the rehabilitation data. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed.

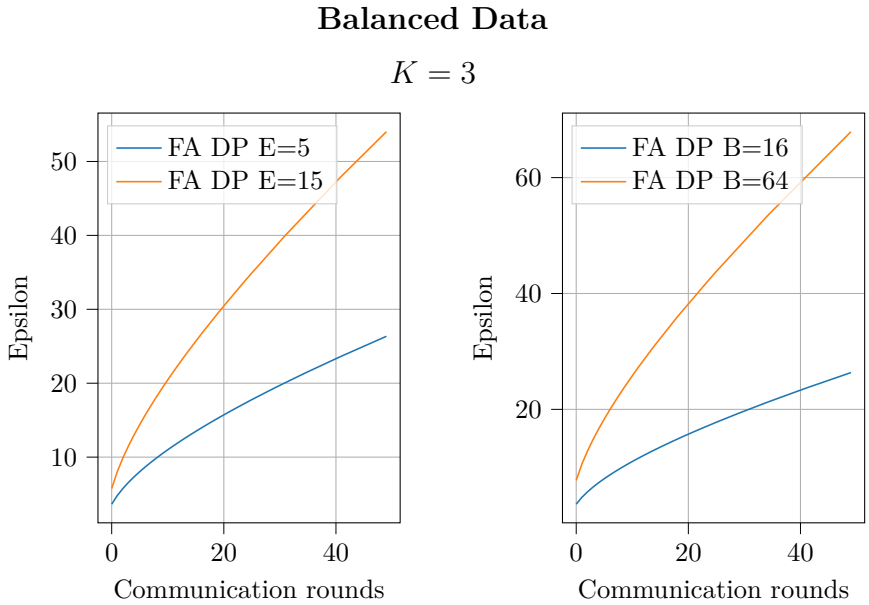


Figure 7.6: Privacy budget ϵ of a global model trained using Federated Averaging with local differential privacy for different epochs and different batch sizes. Training was done using the rehabilitation data. The data was balanced and not label skewed. Batch size was fixed to 16 and number of epochs was fixed to 5, respectively. Number of clients $K = 3$.

7.5 Real Data Experiment 3: Computation Time

7.5.1 Purpose

The purpose of this experiment is to compare the different federated algorithms with respect to computation time.

7.5.2 Results

When varying one parameter the other parameters are set to default: $E = 5$, $B = 16$, and $K = 3$ respectively. This gives some redundancies in the results which have been removed. For this experiment the data distribution is balanced with no label skew. The computation times of Federated Averaging versus Federated Averaging with differential privacy can be seen in Table 7.2 for various epochs E, number of clients K, and batch size B. The time unit for all results are showed in seconds with standard deviations over three runs.

	E		
Algorithm	5	10	15
FedAvg	14.11 ± 0.10	25.23 ± 0.03	36.41 ± 0.03
FedAvg (DP)	46.84 ± 0.21	77.42 ± 0.03	108.57 ± 0.56
Increase	x3.32	x3.07	x2.98
	B		
Algorithm	16	32	64
FedAvg	—	9.22 ± 0.08	7.11 ± 0.01
FedAvg (DP)	—	43.76 ± 0.21	53.35 ± 0.32
Increase	x3.32	x4.75	x7.50
	K		
Algorithm	3	5	10
FedAvg	—	15.00 ± 0.05	17.67 ± 0.15
FedAvg (DP)	—	69.61 ± 0.07	159.63 ± 0.86
Increase	x3.32	x4.64	x9.03

Table 7.2: Computation time of different algorithms and settings by varying number of clients K with epochs $E = 5$ and batch size $B = 16$.

The computation times of Federated Averaging versus Federated Averaging with SMPC can be seen in Table 7.3. Since different environments are used for SMPC and differential privacy, the pure Federated Averaging algorithm is run for both environments.

	E		
Algorithm	5	10	15
FedAvg	172.08 ± 1.83	339.80 ± 2.49	506.30 ± 6.44
FedAvg (SMPC)	174.41 ± 0.55	342.73 ± 1.10	511.16 ± 0.43
Increase	x1.01	x1.01	x1.01
	B		
Algorithm	16	32	64
FedAvg	—	88.88 ± 0.15	51.14 ± 0.25
FedAvg (SMPC)	—	91.14 ± 0.93	53.14 ± 0.09
Increase	x1.01	x1.03	x1.04
	K		
Algorithm	3	5	10
FedAvg	—	175.70 ± 1.73	193.78 ± 0.36
FedAvg (SMPC)	—	180.55 ± 0.99	209.03 ± 3.58
Increase	x1.01	x1.03	x1.08

Table 7.3: Computation time of different algorithms and settings by varying number of clients K with epochs $E = 5$ and batch size $B = 16$.

Algorithm	$B = 16$		
Central	7.97 ± 0.17		
	E		
Algorithm	5	10	15
FA (PT1.4)	x21.58	x42.61	x63.49
FA (PT1.7)	x1.77	x3.16	x4.57
	B		
Algorithm	16	32	64
FA (PT1.4)	x21.58	x11.14	x6.41
FA (PT1.7)	x1.77	x1.16	x0.89
	K		
Algorithm	3	5	10
FA (PT1.4)	x21.58	x22.03	x24.30
FA (PT1.7)	x1.77	x1.88	x2.22

Table 7.4: Increase in computation time of Federated Averaging for various numbers of epochs E, batch size B, and clients K compared to a centrally trained model where $B = 16$.

7.5.3 Discussion

The experiment results in Table 7.2 show that the differentially private Federated Averaging algorithm requires more than three times the computation time of the pure Federated Averaging algorithm. This ratio grows when increasing either B and K. When increasing epochs E the ratio slightly decreases however.

In the case of SMPC in Table 7.3, the increase in computation is roughly the same as the pure Federated Averaging algorithm which indicates that the SMPC algorithm is very performant when varying the different parameters. However, it should be noted that the values shown in the tables above do not account for the communication delays of Federated Averaging. Since the computation was carried out on a single compute node where clients were processed sequentially, and the data was kept constant even for increasing numbers of clients, these timings for various values of K may be misleading. In a real federated system, increasing the number of clients (with similar round times) should not have an impact on the pure Federated Averaging algorithm since it operates in a lockstep, or synchronized, fashion where the next round is only started when all clients have responded to the server with their model update.

Finally, in Table 7.4 the computation times of the federated averaging algorithms are compared to the central case where $B = 16$. We see that the use of Federated Averaging with PyTorch 1.4 (including PySyft with VirtualWorkers for SMPC) increases computation time much greater than the case where pure PyTorch 1.7 is used (used for differential privacy).

7.6 Real Data Experiment 4: Individual Client Accuracy with Local Models

7.6.1 Purpose

In this experiment the clients each have a locally trained neural network that is not aggregated at any point. The purpose of this experiment is to understand how a global model trained using Federated Averaging performs in relation to these locally trained models under various data distributions. The centrally trained model is again used as overall baseline.

7.6.2 Results & Discussions

The data can be balanced or unbalanced such that some clients have much more data than others. Additionally, the label distribution can be even or skewed such that some clients have more of certain labels and less of others. This gives four combinations of data distributions that are experimented with. Each line in the plots of this section corresponds to a single client, and different colors represent different parameters or learning approaches.

Balanced Data

In this case the data is balanced across clients and the label distribution is even. Figure 7.7 shows that all models perform better in training accuracy compared to the centrally trained baseline (blue). For both training and testing accuracy a higher number of epochs where $E = 15$ performs better than $E = 5$ for both Federated Averaging and the local models. In test accuracy however, all the models generally converge to the same plateau as the central model. We also see that some clients with the federated models (green and yellow) reach higher accuracies than other models with a difference of about 8 to 10 percentage points. For two of the clients, the federated algorithm with $E = 15$ (green) reaches the same accuracy as the central model but in less than 20 communication rounds. The local model with $E = 15$ (purple) roughly reaches the same plateau as the central model with the exception of one client that seems to diverge in the end.

When varying batch size in Figure 7.8 a lower batch size of $B = 16$ performs better for both federated and local models. It can also be seen that the federated models (green and yellow) perform better than their non-federated counterparts (red and purple).

In Figure 7.9 it can be seen that the models with higher epochs $E = 15$ (green and purple) reach high AUC scores in both training and test. However, when increasing the batch size, the AUC scores fall well below the baseline for both federated and local models.

The precision and recall values can be seen in Figure B.4 and Figure B.5 in the appendix.

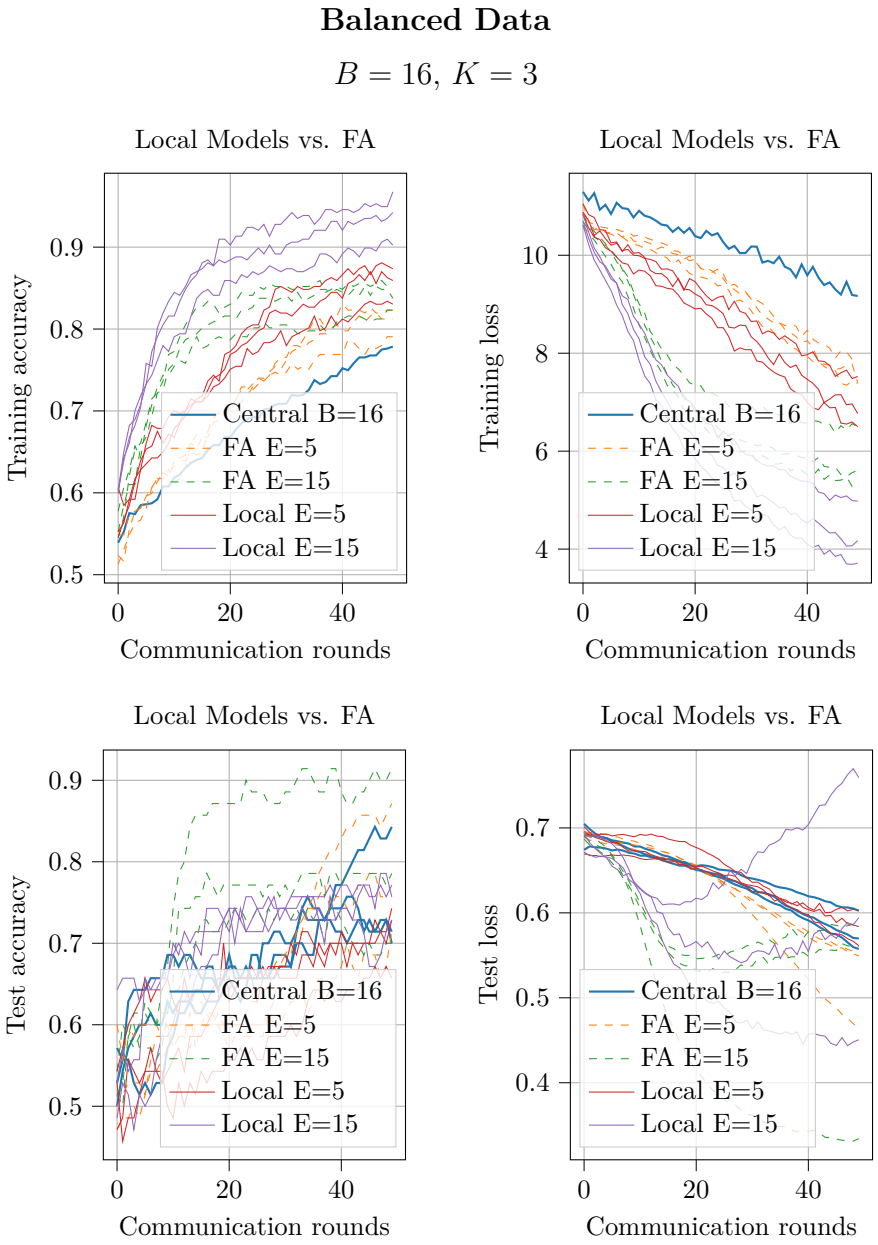


Figure 7.7: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

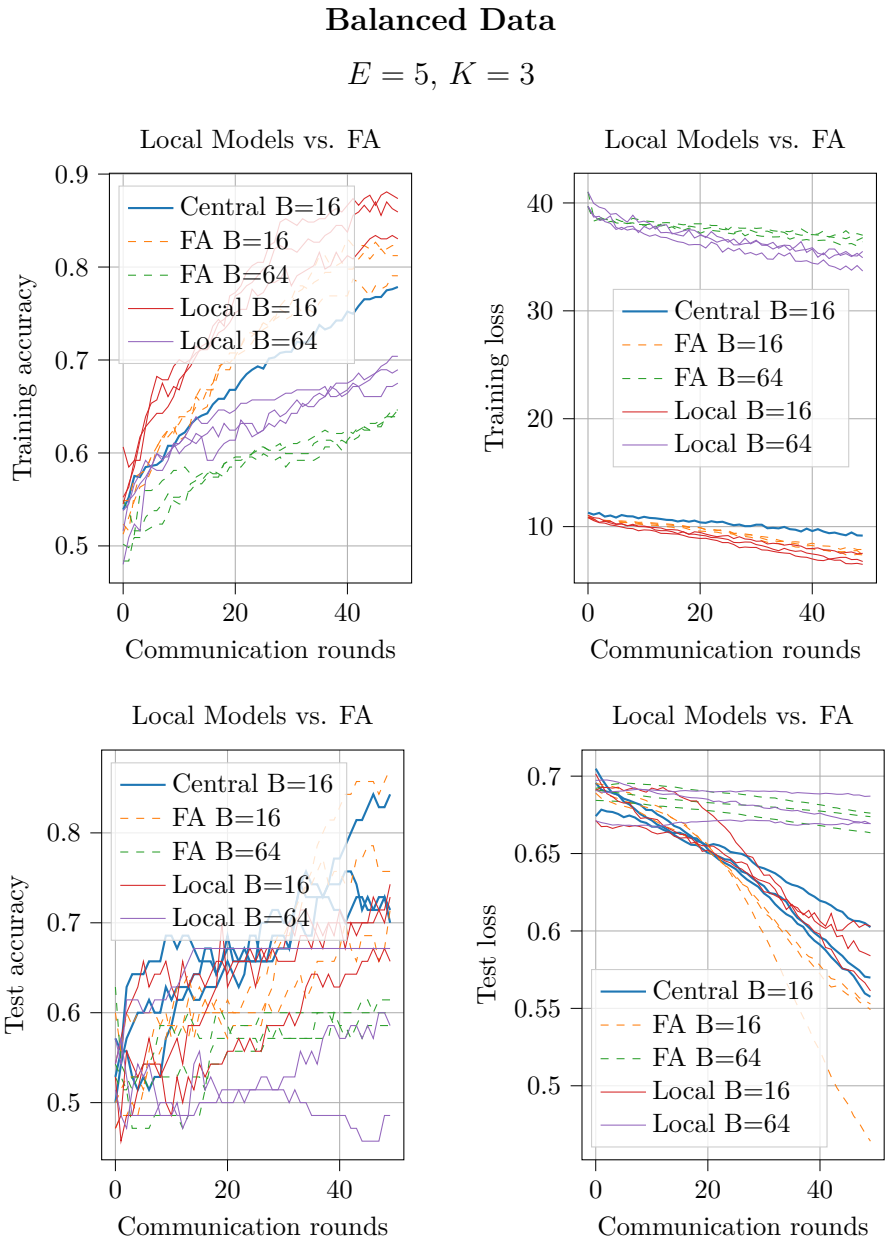


Figure 7.8: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

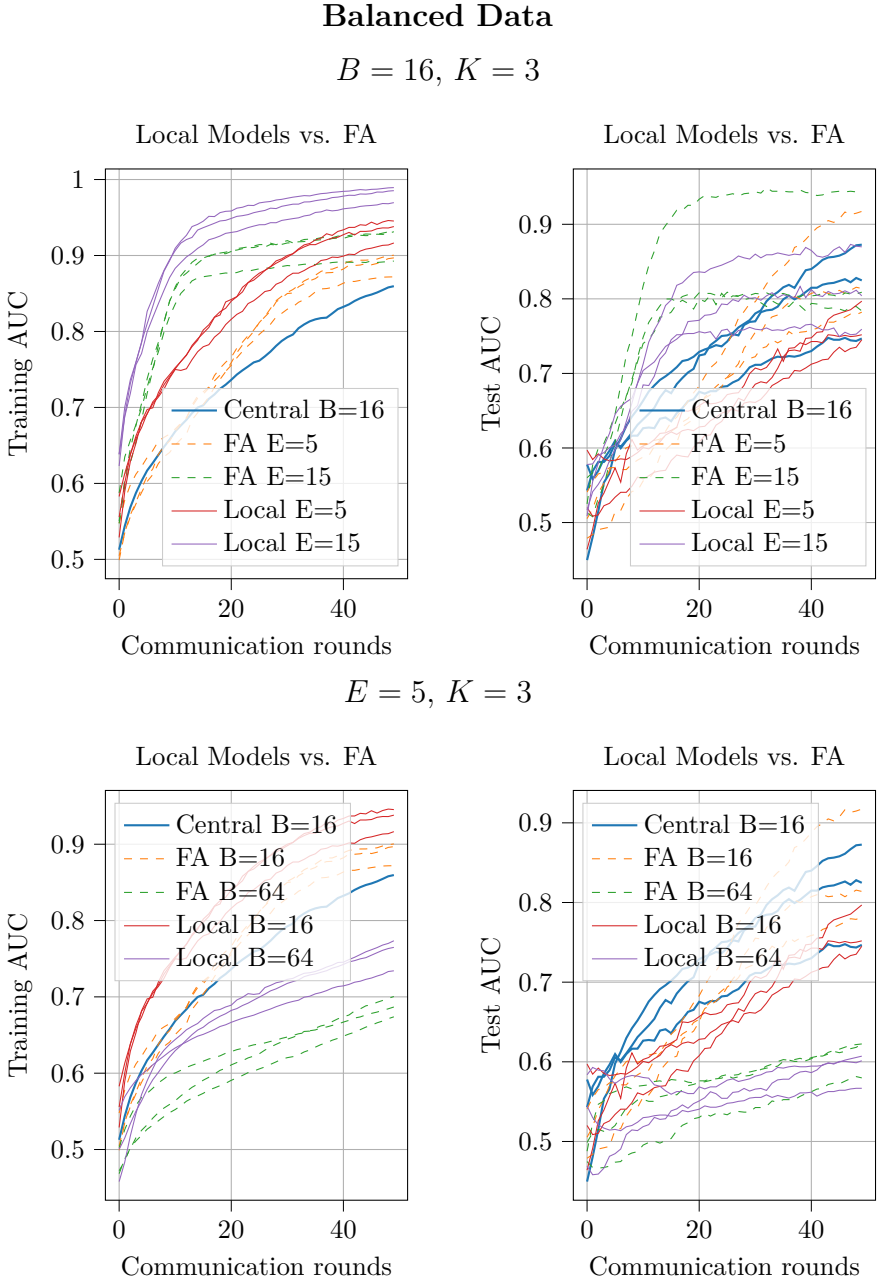


Figure 7.9: AUC during both training and testing of local models trained for different batch sizes. Training was done using the rehabilitation data. The data was balanced and not label skewed.

Unbalanced Data

In this case the data is unbalanced across clients and the label distribution is even. For test accuracy, we see the same pattern as with the MNIST data. The local models do not perform well on all clients if the clients have unbalanced amounts of data. Looking in the experiment logs, the lines with low accuracy on Figure 7.10 and Figure 7.11 indeed correspond to the clients with little data. These clients perform well with regards to training accuracy but fails to generalize as seen in the test accuracies. On the other hand, we see that Federated Averaging in general performs better than the local models in test when varying either epochs or batch size. This indicates that models trained on clients with more data can learn patterns that are beneficial for clients with less data when the data is without label distribution skew.

Finally, for the unbalanced data set we see in Figure 7.12, that the clients with a local model achieve high training AUC - surpassing the federated models for various epochs. Though, some of these local models drop significantly in test AUC, whereas the federated models achieve performance close to the central baseline or better. We see the same pattern for various batch sizes, although, a higher batch size reduces the AUC below the baseline.

The precision and recall values can be seen in Figure B.6 and Figure B.7 in the appendix.

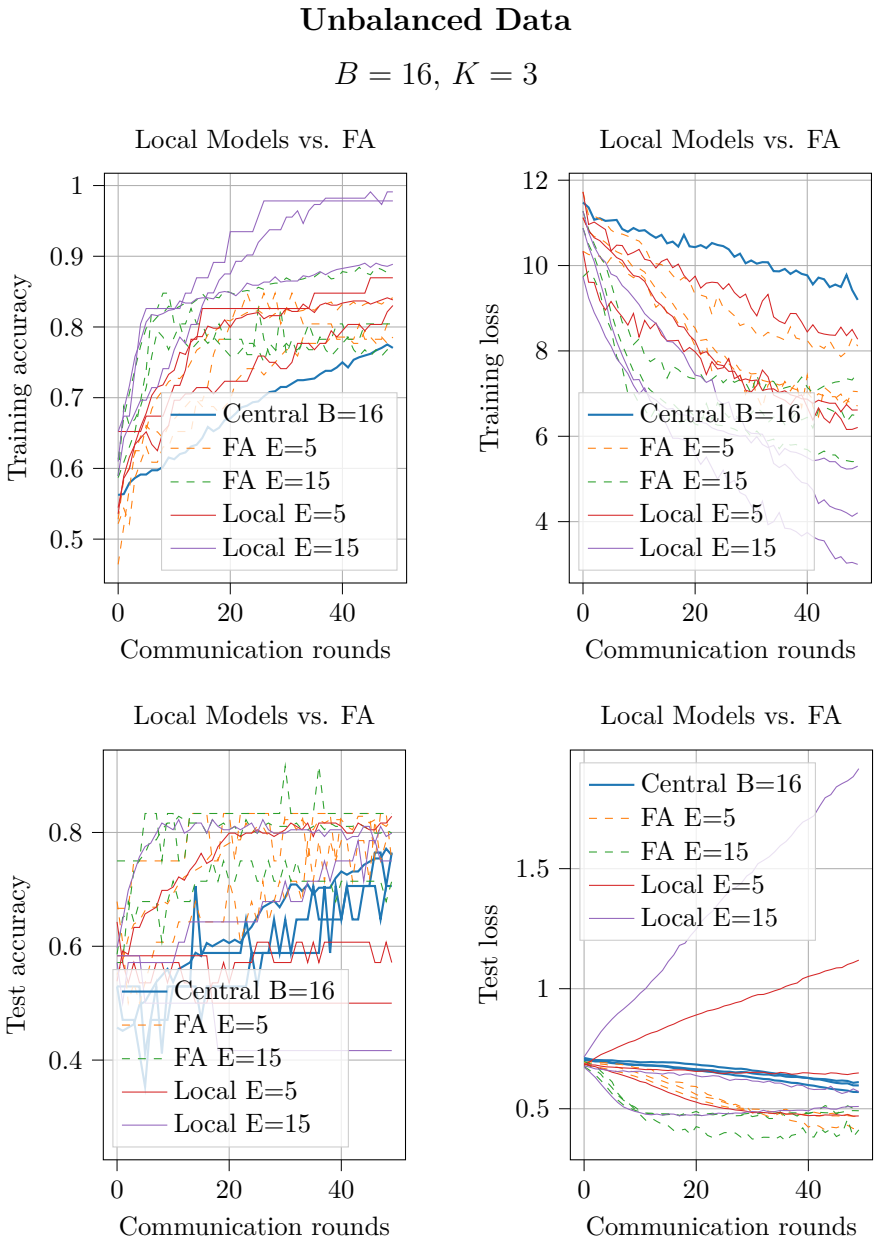


Figure 7.10: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

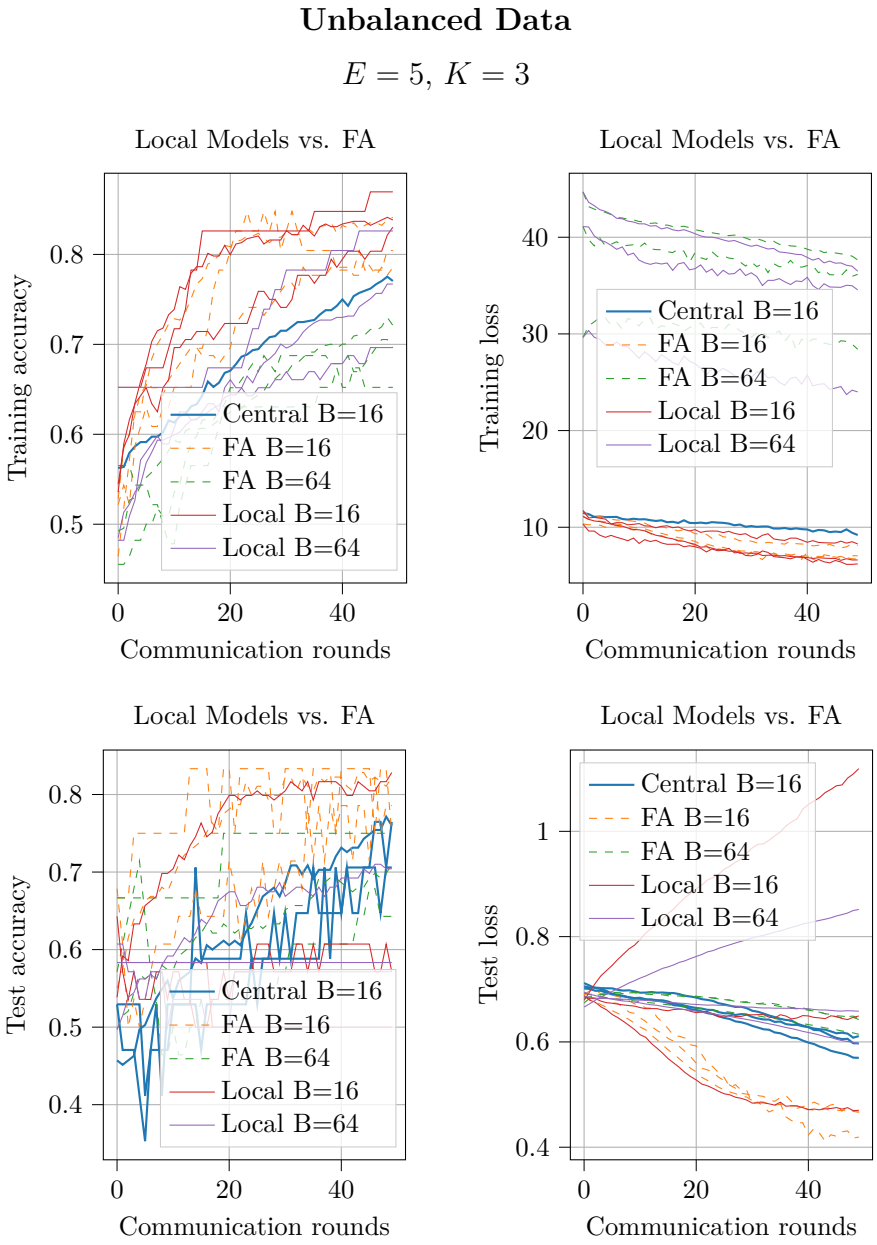


Figure 7.11: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

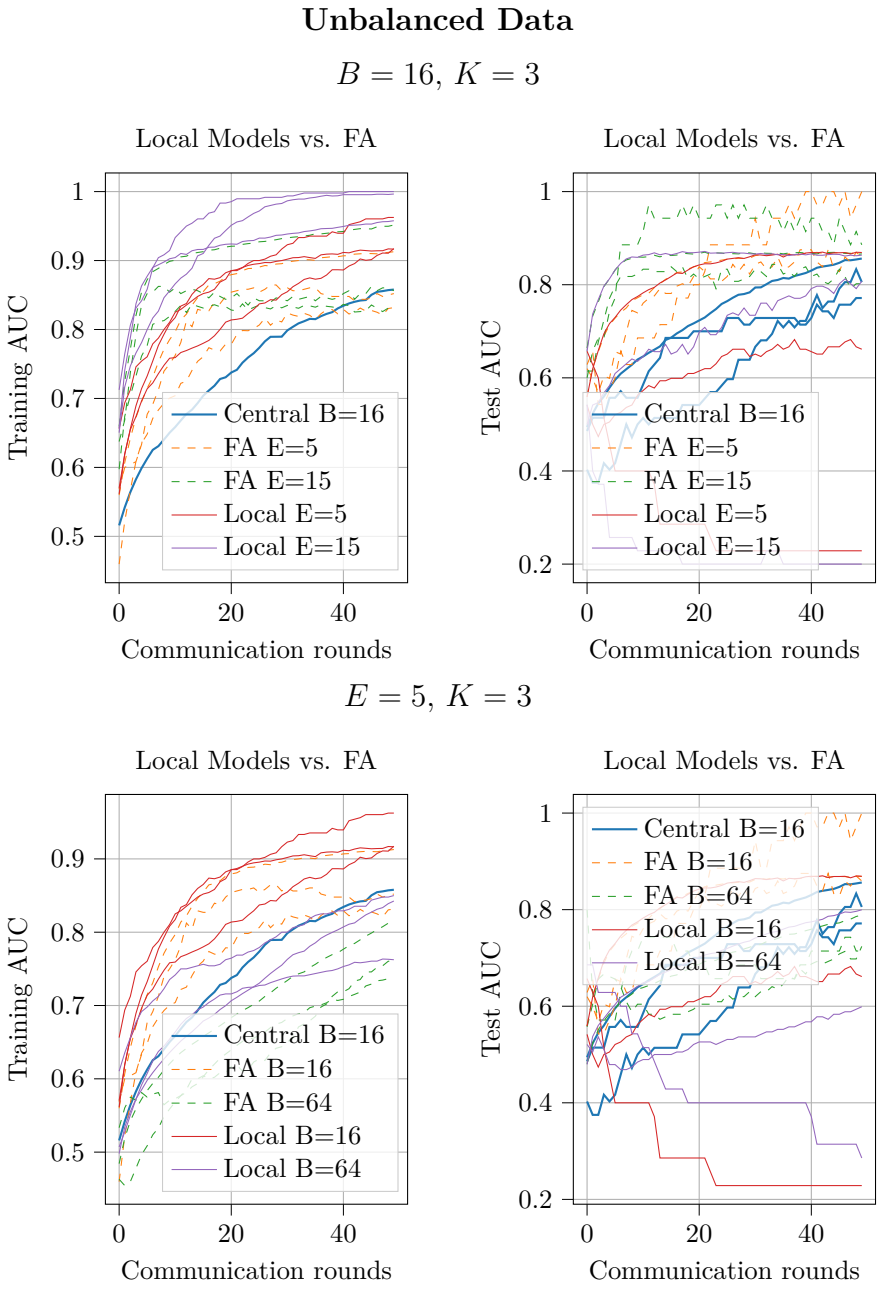


Figure 7.12: AUC during both training and testing of local models trained for different batch sizes. Training was done using the rehabilitation data. The data was unbalanced and not label skewed.

Balanced Data with Label Distribution Skew

In this case we see in Figure 7.13 that the federated models (yellow and green) generally achieve higher accuracies than both the centrally trained baseline (blue) and the local models (red and purple) in test. The federated models seem to generalize better than the other models when comparing training and testing results, and specifically when $E = 15$ the number of communication rounds is greatly reduced. The local models generally achieve better test accuracy than the baseline. In Figure 7.14 we see roughly the same trends, however, the a lower batch size reaches better accuracy for both local and federated models.

The AUC scores can be seen in Figure 7.15 where both the local and federated models perform better in training than the baseline for various epochs. In test AUC however, the federated models are able to reach higher scores in less rounds than the local models and the baseline. The AUC scores during both training and testing drops significantly when increasing batch size to $B = 64$, but the federated model is still able to reach a higher AUC than the baseline in both cases.

The precision and recall values can be seen in Figure B.8 and Figure B.9 in the appendix.

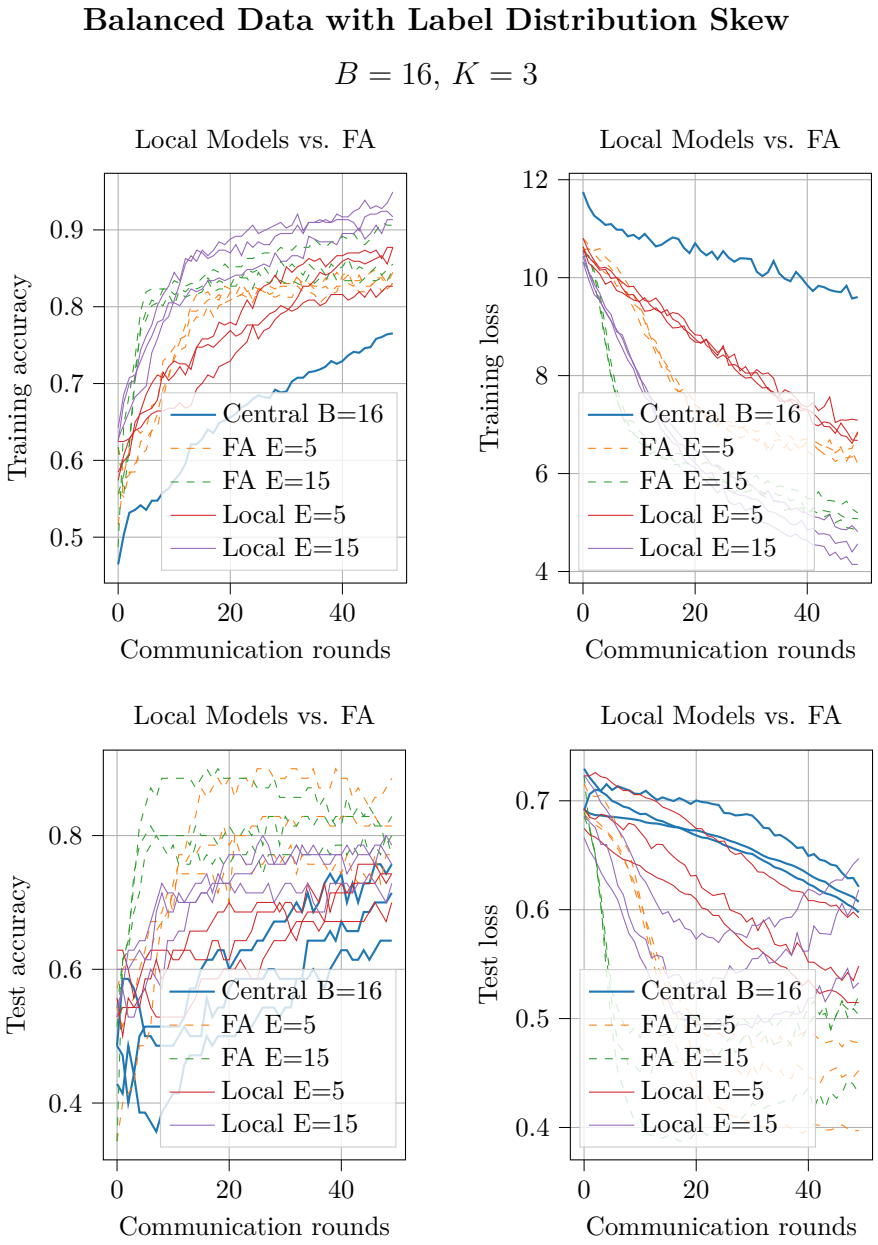


Figure 7.13: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

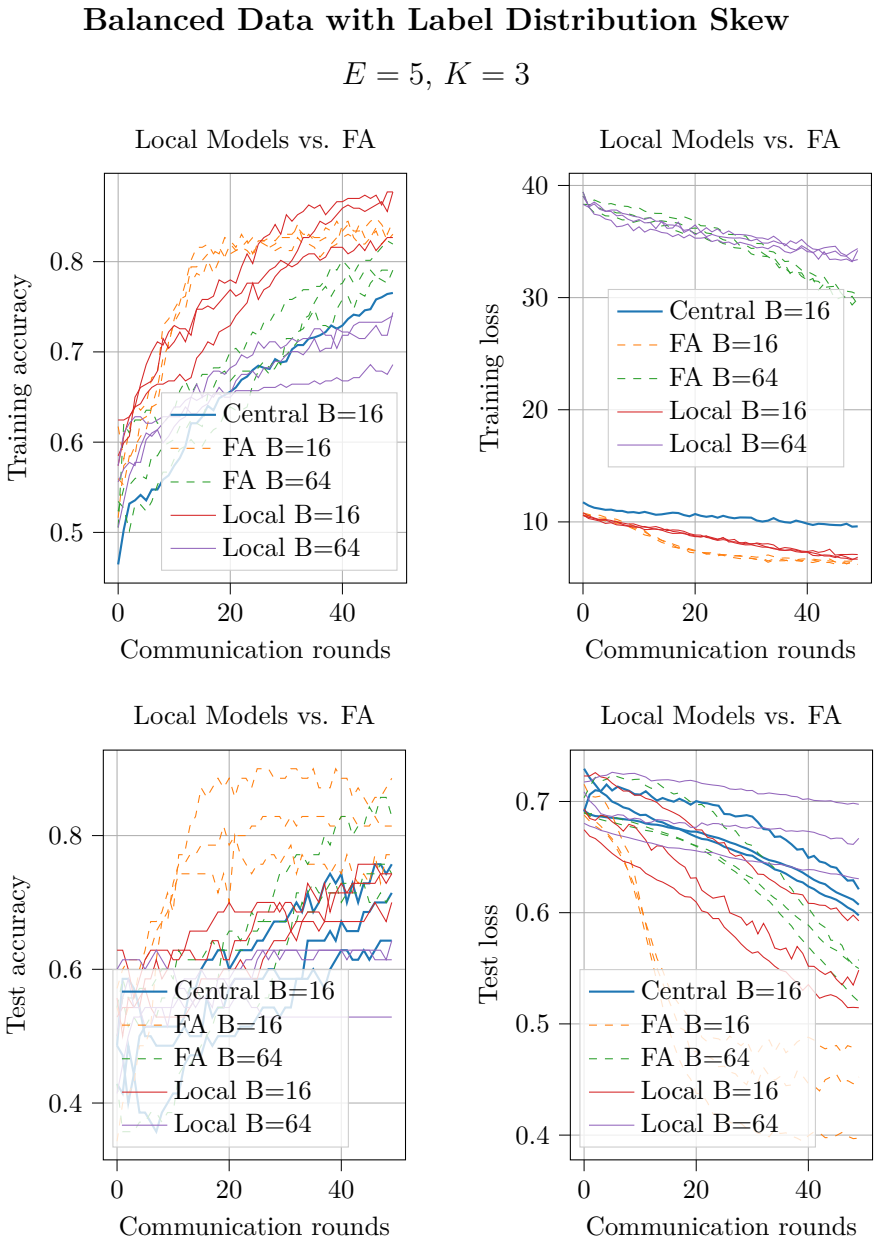


Figure 7.14: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

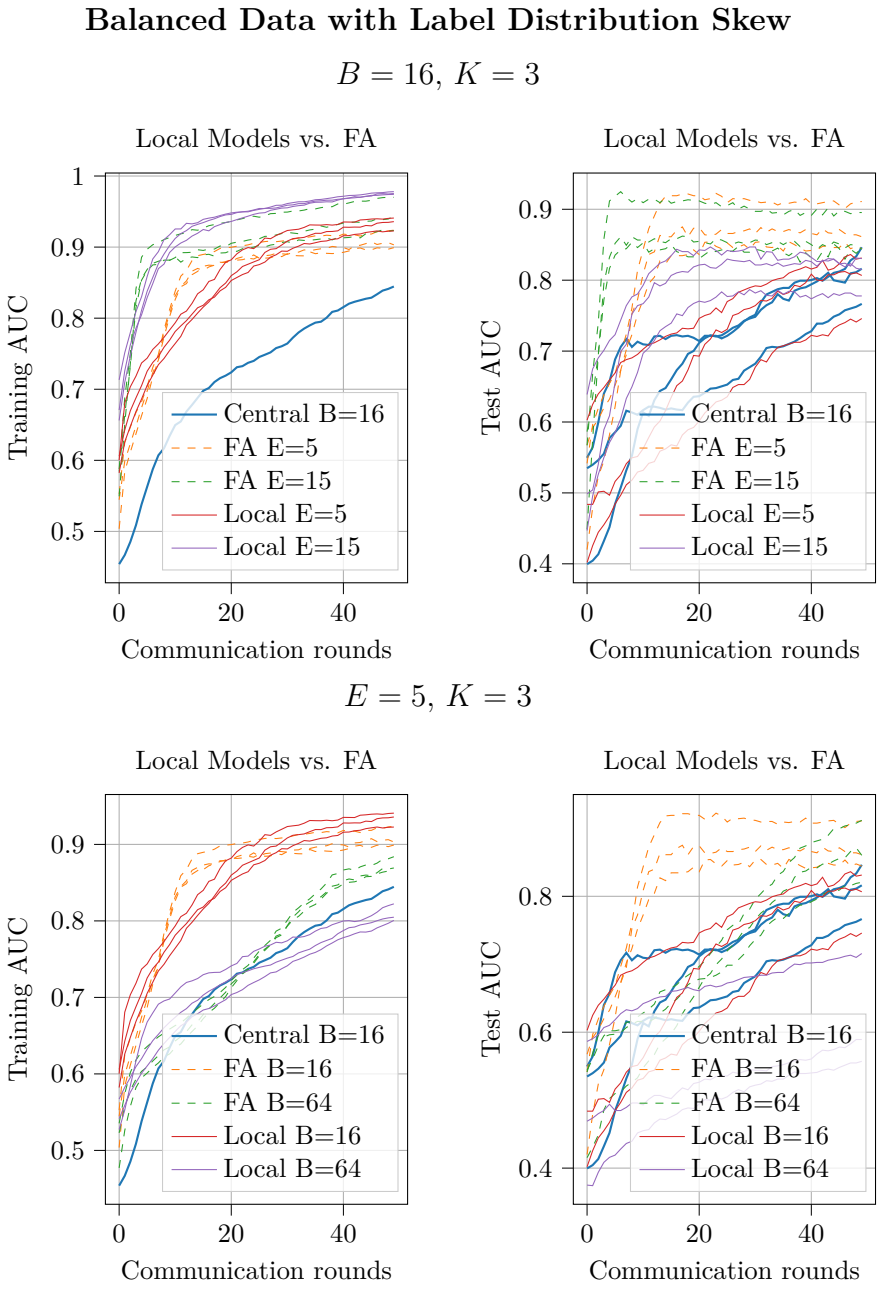


Figure 7.15: AUC during both training and testing of local models trained for different batch sizes. Training was done using the rehabilitation data. The data was balanced and label skewed.

Unbalanced Data with Label Distribution Skew

We see rather high variation in test accuracy between clients of all types of models, including the centrally trained non-federated baseline, in Figure 7.16. Despite this, most of the models tend to end up with about 70% to 80% test accuracy. One of the clients of the central model provides the lowest test accuracy of about 60%. In Figure 7.17 we see that increasing the batch size for the federated model (green) increases the number of communication rounds needed to reach the same accuracies as the federated model with low batch size during both training and testing.

The locally trained models in Figure 7.18 achieve high training AUC scores that surpass both the federated models and the non-federated model for various epochs. However, their performance drops in test AUC where a few clients reach only about 60% which is on par with the lowest-performing client of the central model. Furthermore, it can be seen that the federated models perform well for all clients with AUC scores of roughly 90%. The results show that increasing batch size to $B = 64$ increases the communication rounds needed for the federated models to reach about the same test AUC when comparing with the federated model where $E = 15$. Likewise for the local models, increasing the batch size reduces the AUC by about 20 percentage points.

The precision and recall values can be seen in Figure B.10 and Figure B.11 in the appendix.

A summary of performance in all four data distribution cases can be found in Table 7.5.

	Rehabilitation Data Set
Data Distribution	Performance Summary
Balanced	The test AUC scores for the federated and local models generally converge to the same plateau as the non-federated model at about 80%.
Unbalanced	The federated models achieve performance that is generally higher than the non-federated case. As with MNIST, the locally trained models struggle with getting high AUC scores on clients with low amounts of data. The training graphs indicate overfitting of these clients.
Balanced with label distribution skew	As opposed to the results from the MNIST data set, here the federated models are able to reach higher test AUC scores more consistently and in fewer rounds than the local models and the non-federated model.
Unbalanced with label distribution skew	Again, the federated models are able to reach higher test AUC scores more consistently and in fewer rounds than the local models and the non-federated model.

Table 7.5: Summary of results regarding Federated Averaging on different data distributions of the rehabilitation data set.

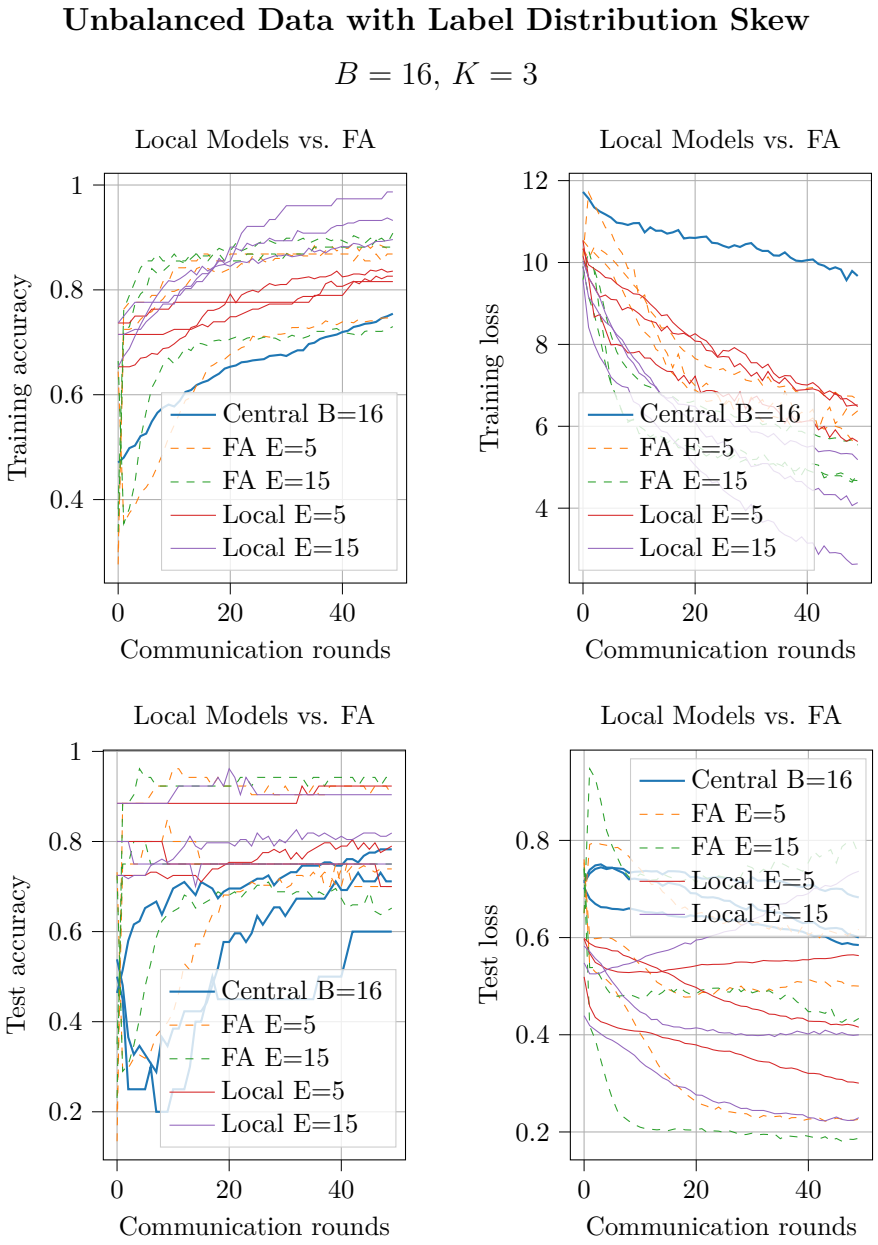


Figure 7.16: Accuracy and loss during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

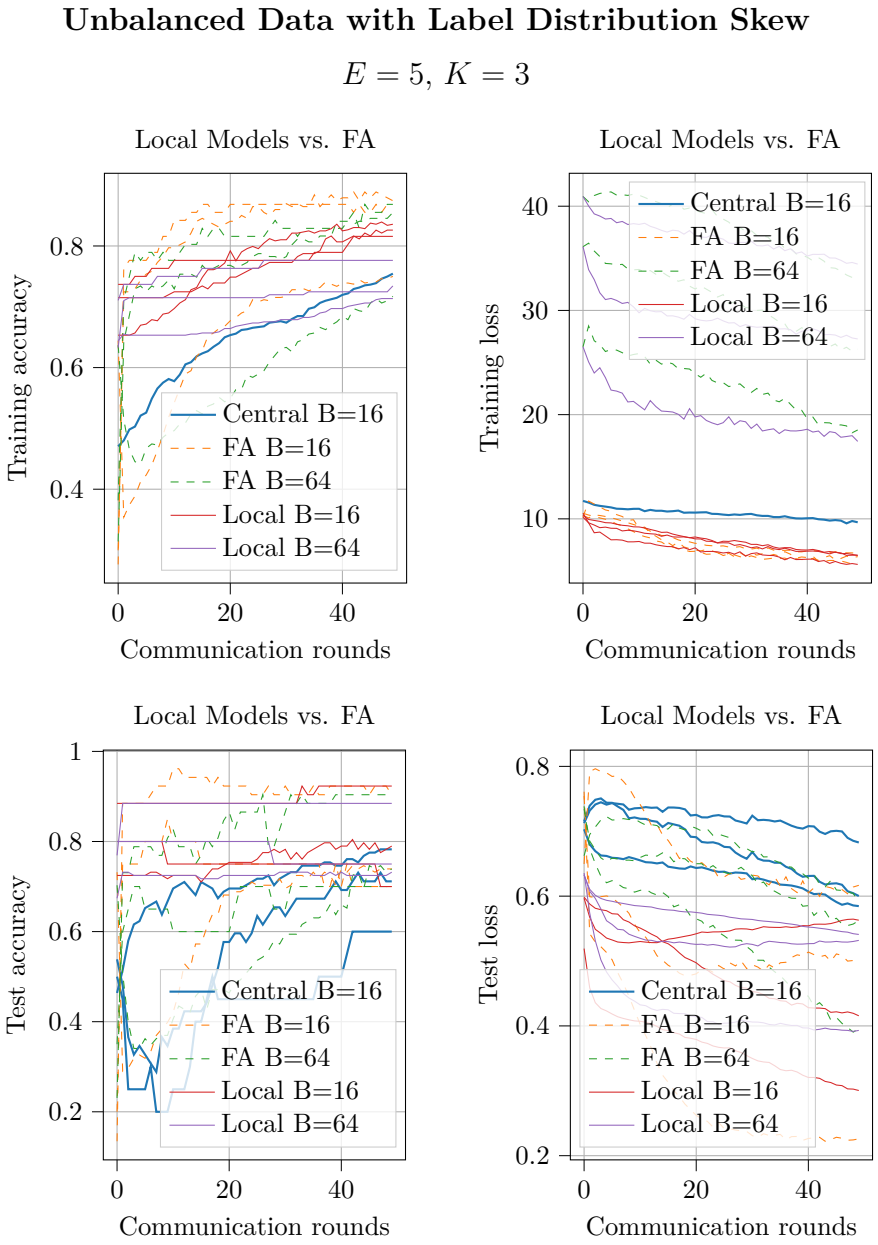


Figure 7.17: Accuracy and loss during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

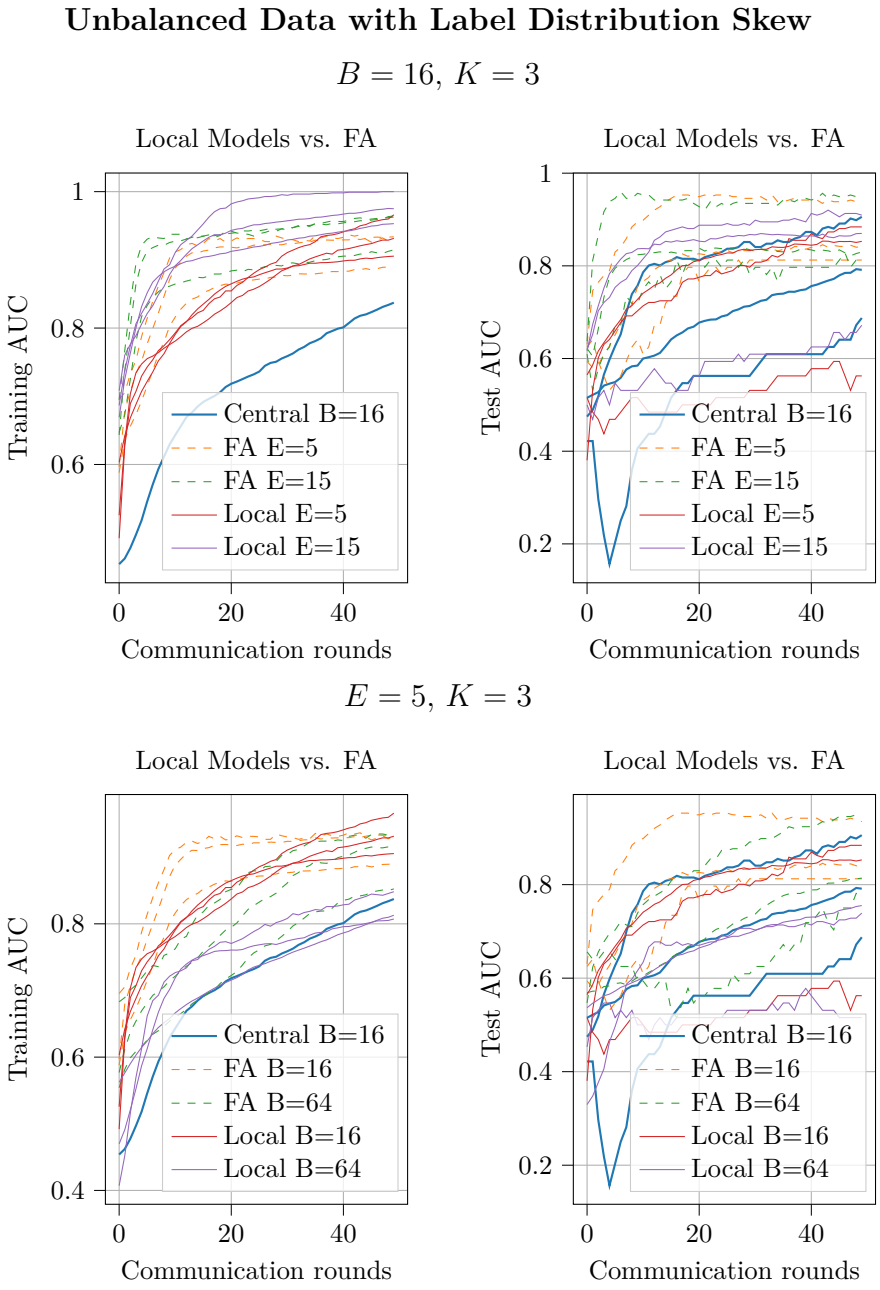


Figure 7.18: AUC during both training and testing of local models trained for different batch sizes. Training was done using the rehabilitation data. The data was unbalanced and label skewed.

7.7 Real Data Experiment 5: Explainability

7.7.1 Purpose

The purpose of this experiment is to see how significantly the explainability changes when changing from a non-federated data-centralized learning approach to a federated approach using Federated Averaging.

7.7.2 Results

A SHAP decision plot of model trained using centralized data is shown in Figure 7.19. Recall that the To allow direct comparison, the feature order on the vertical axis is fixed for all decision plots. In Figure 7.20 two SHAP decision plots are shown using models trained using Federated Averaging with and without differential privacy.

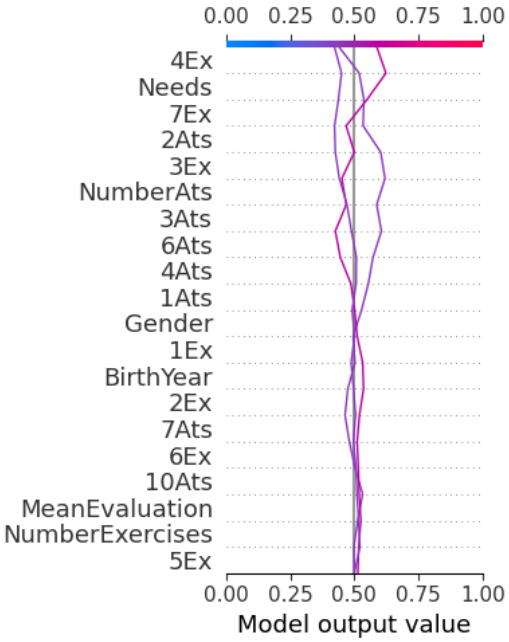
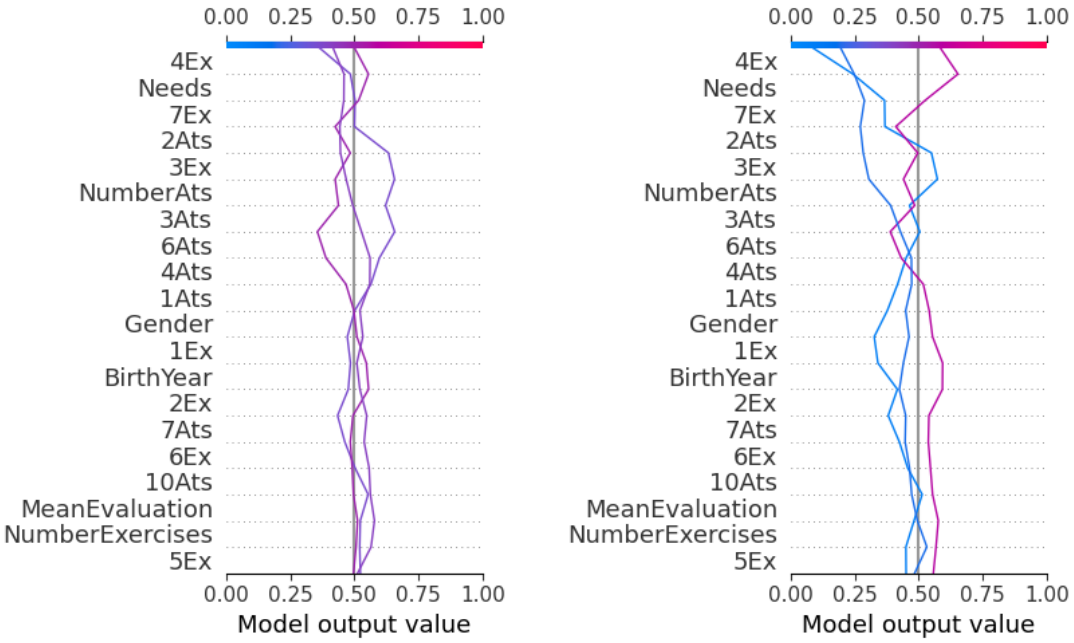


Figure 7.19: SHAP decision plot for a non-federated model trained with centralized rehabilitation data. The plot shows the most important features in decreasing order along the y-axis for 3 predictions.

7.7.3 Discussion

At first glance the plots in Figure 7.20 seem rather different from the decision plot of the non-federated model in Figure 7.19. On closer inspection however, the lines of both Federated Averaging methods (with and without differential privacy) seem to merely have been shifted or skewed slightly when comparing with the decision plot of the non-federated model. We also



(a) Federated Averaging $E = 5$, $B = 16$, and $K = 3$.

(b) Federated Averaging with differential privacy $E = 5$, $B = 16$, and $K = 3$.

Figure 7.20: SHAP image plots using models trained with (a) Federated Averaging and (b) Federated Averaging with differential privacy on rehabilitation data. The plot shows the most important features in decreasing order along the y-axis for 3 predictions.

observe that the decision plot lines of the differentially private method are more shifted and skewed than the pure Federated Averaging algorithm.

Chapter 8

Discussion & Conclusion

In a conventional machine learning setting, the need for large amounts of data to be centralized poses issues with respect to privacy preservation. This is especially a concern due to regulations such as the General Data Protection Regulation in EU. This thesis examines federated learning, a different machine learning approach that seeks to train a privacy-preserving model on decentralized data without requiring sensitive data to be shared. In this work, artificial neural networks were employed as these often provide state of the art performance in the industry for many different tasks. The models have been trained using three different federated algorithms that were subject to experimentation during this work. A traditional non-federated data-centralized model with the same neural network architecture has also been trained and used as baseline for comparison. The federated algorithms are:

- Federated Averaging.
- Federated Averaging with Differentially Private Stochastic Gradient Descent.
- Federated Averaging with Secure Multi-Party Computation.

The differentially private algorithm provides local differential privacy. Local differential privacy provides the best privacy guarantees in a federated setting because clients do not need to trust any central server or other clients, as opposed to global differential privacy. The results from the differential privacy experiments on a reduced set of the MNIST data set show that the differentially private algorithm reduces the accuracy of about 15 percentage points compared to either the non-federated model or the federated model without differential privacy. In this case, the pure Federated Averaging algorithm, with the most performing hyperparameters, performed at the same level as the non-federated model. For the rehabilitation data set, the pure Federated Averaging algorithm is able to reach an AUC score roughly equal to the non-federated model in an optimal data distribution case. The differentially private algorithm also reaches the same AUC but requires more communication rounds.

The resulting privacy budget ϵ is higher when training using the rehabilitation data set than for MNIST. To reduce the privacy budget, one can reduce the batch size B and the number of local epochs E. Due to the applied differentially private algorithm, the data set sizes can influence the privacy budget greatly, and the privacy budgets may be reduced if a larger data set can be formed. Another observation regarding differential privacy is that the

added noise may act as a regularizer. An excessive amount of regularization may reduce the performance of the differentially private algorithm. In this case removing dropout is an option to be explored.

Secure Multi-Party Computation (SMPC) has been applied to Federated Averaging to protect the clients' model weights against an untrusted aggregator, i.e. the central server performing the weight aggregation. This is one of the main reasons to apply SMPC in a federated setting. The results of these experiments show that using SMPC with Federated Averaging does not substantially affect the results of the pure Federated Averaging algorithm. This is likely due to the fact that the precision of the fixed precision encoding is set sufficiently high such that there is very little or no difference in the model weights after encoding.

The wall clock time was also measured of all algorithms on both the MNIST data set and the rehabilitation data set during training. The experiments do not take into account the communication delay, but rather only the computation time. Furthermore, the measurements include calculation times of both the server and the clients, meaning that it can not be directly determined how much each party contributed. The results show that the differentially private federated algorithm performs at least five times slower than the pure Federated Averaging algorithm on MNIST, and at least three times slower on the rehabilitation data for the considered parameters. The results indicate that the calculation of additive secret shares for SMPC take almost no additional time on either data set. The computation times were also compared to the non-federated case where it was found that using the Python library PySyft (library for private federated learning) has a large impact on computation time. The increase was up to 82.5 times the central case on MNIST, whereas the environment without PySyft was only a 6-fold increase for the same hyperparameter settings. It was found that the non-federated data-centralized model is much faster to train than the other models.

The different algorithms have been examined in four different data distribution cases. The data can be balanced or unbalanced such that some clients have much more data than others. Additionally, the label distribution can be even or skewed such that some clients have more of certain labels and less of others. This gives four combinations of data distributions that have been experimented with.

For the MNIST data set, the federated models in general reach accuracies similar to the non-federated baseline for three out of four data distribution cases. When the data is balanced with label distribution skew, some clients will have relatively low accuracy. This may be due to the fact that these clients have high label distribution skew between samples available to the clients. For the rehabilitation data set, the federated models in general reach the same AUC scores or higher compared to the non-federated baseline for all the four data distribution cases. It was found for both data sets that a global model trained using Federated Averaging in general performs better than locally trained models for the four different data distributions. Furthermore, the precision and recall curves for the federated models are generally comparable or better than the non-federated model. Overall, the results indicate that averaging the neural network weights using Federated Averaging can have a positive effect on the performance of all clients.

Finally explainability was experimented with, where it was found that SHAP is able to explain the neural networks used in this thesis. The results also show that the feature con-

tributions do not change significantly on the rehabilitation data set across a non-federated data-centralized model, Federated Averaging, and Federated Averaging with Differentially Private Stochastic Gradient Descent.

8.1 Comparison with State of the Art

We see that the privacy preserving algorithm of our experiments on the rehabilitation data achieves about (17, 10^{-5})-differential privacy for the best-performing hyperparameters. This performance result can be compared to the recurrent language model of McMahan et al. [33]. The authors show, in a cross-device setting, that their differentially private algorithm achieves (4.6, 10^{-9})-differential privacy with no significant decrease in model accuracy given a data set with 763,430 users. We see that the privacy budget is much lower in comparison. Although the privacy budget ϵ is high on the rehabilitation data set, the algorithm still provides additional privacy on top of the Federated Averaging algorithm while achieving comparable AUC scores to the non-private federated algorithm.

8.2 Contributions

1. Predictive performance has been compared between three federated learning algorithms and a conventional data-centralized model on four different data distributions across clients for two different data sets.
2. The feasibility of using SHAP as a method for explaining a neural network has been investigated along with the influence that the federated models could have on explanations.
3. Comparisons have been made of the computation times of applying the three algorithms.
4. A framework has been created to support experimentation of: conventional data-centralized models, federated models with either secure multi-party computation or local differential privacy, and four different cases of data distribution. Additionally, a framework for plotting the results of these experiments has also been created.

8.3 Conclusion

In this thesis, we have documented and described how to design, implement, and test the Federated Averaging algorithm and variants thereof for training neural networks on decentralized data. Three algorithms were considered: pure Federated Averaging, Federated Averaging with Differentially Private Stochastic Gradient Descent, and Federated Learning with Secure Multi-Party Computation. It has been shown experimentally that the algorithms could learn and predict on a benchmarking data set with good overall performance. These algorithms are subsequently evaluated on a data set regarding physiotherapy-based rehabilitation performed in municipalities of Denmark. The results indicate that Federated Averaging generally performs

well on four different data distributions for both data sets. Additionally, the AUC scores on the rehabilitation data set are comparable or better than the conventional data-centralized model and locally trained models on the clients. Furthermore, the precision and recall curves for the federated models are generally comparable or better than the non-federated model. The differentially private algorithm performs with significantly lower accuracy compared to Federated Averaging on MNIST, but reaches the same AUC score on the rehabilitation data. The computation time of the differentially private algorithm is significantly higher than for the algorithm with SMPC (about a factor of 3 on the rehabilitation data). Furthermore, SMPC has no significant influence on model weights for high enough precision on the fixed precision encoding. Finally, the results show that the SHAP explanations are similar across the different algorithms.

8.4 Future Work

The use of accuracy as a metric for predictive performance on data with label imbalance may have skewed the results of the MNIST experiments, i.e. two out of the four cases in experiment 5. Future work should involve using a more fitting metric on these cases to verify validity.

The computation time of the different algorithms were measured using a single compute node. For future work, it would be interesting to experiment with multiple nodes in a parallel setup to investigate the effect of communication delays and unreliable clients.

For better performance on the rehabilitation data set, further hyperparameter optimization should be performed. Techniques such as Grid Search or Random Search could be applied to find e.g. more suitable learning rates, model architectures, and regularization techniques.

Furthermore, experiments were conducted of local differential privacy which provides the most private federated setting, however, no formal argumentation of the privacy guarantees have been made for this particular algorithm. This should be done before applying the algorithm in other applications. Finally, it may be worthwhile to investigate global differential privacy in an attempt to attain better predictive performance while keeping or improving privacy guarantees. As the setting of physiotherapy-based rehabilitation is characterized by a significant amount of trust between municipalities, it would be sensible to take advantage of this fact. Another interesting direction to investigate could be application of differential privacy in a non-federated data-centralized setting.

Appendix A

Confusion Matrices

A.1 Non-Federated Model on MNIST

A confusion matrix for $B = 16$ is shown in Figure A.1 where it can be seen that most values lie on the diagonal. The confusion matrix indicates that the model has achieved good performance for all of the 10 classes.

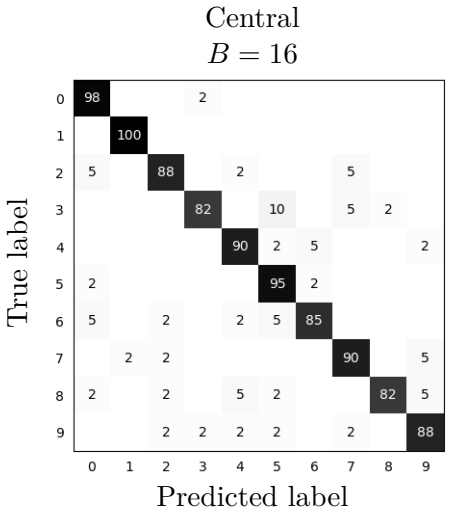


Figure A.1: Confusion matrix of a centralized model trained using centralized MNIST data with batch size $B = 16$. The values of the matrix are shown in percentages for each row.

A.2 Non-Federated Model on Rehabilitation Data

A confusion matrix for $B = 16$ is shown in Figure A.2 where it can be seen that most values lie on the diagonal. The confusion matrix indicates that the model has achieved somewhat good performance where the highest values lie on the diagonal. Most of the errors we see of this model are false negatives.

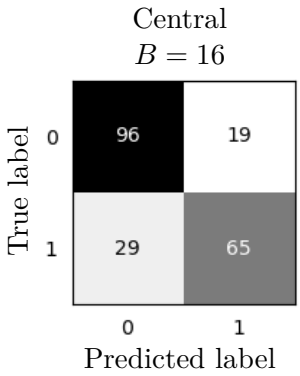


Figure A.2: Confusion matrix of a centralized model on centralized rehabilitation data trained with batch size $B = 16$.

Appendix B

Precision & Recall

B.1 Real Data Experiment 1: Accuracy with Centralized Model and Data

The precision and recall of the centralized model can be seen in Figure B.1. We see here that the model with $B = 16$ performs better in both precision and recall.

B.2 Real Data Experiment 2: Accuracy with Local Differential Privacy

For the precision and accuracy scores in Figure B.2 we see that increasing the number of epochs to $E = 15$ (red and purple) leads to better scores with and without differential privacy. The precision of Federated Averaging with $E = 15$ without differential privacy (red) reaches about the same plateau as the central model in less than 20 communication rounds. The differentially private version of $E = 15$ (purple) similarly reaches the same plateau in test precision after roughly 20 communication rounds. When varying the batch size in Figure B.3 we see that low values of $B = 16$ perform better than $B = 64$. Furthermore, the differentially private versions again achieve lower scores overall.

B.3 Real Data Experiment 4: Individual Client Accuracy with Local Models

B.3.1 Balanced Data

In terms of precision, in Figure B.4, most models seem to perform slightly worse than the central baseline. Though, a few of the clients (green, yellow, and purple) quickly reach high precision scores, but the other clients do not follow. For the recall scores, however, the federated models and the local model with $E = 15$ (green, yellow, and purple) overtake the baseline with about 5 to 10 percentage points.

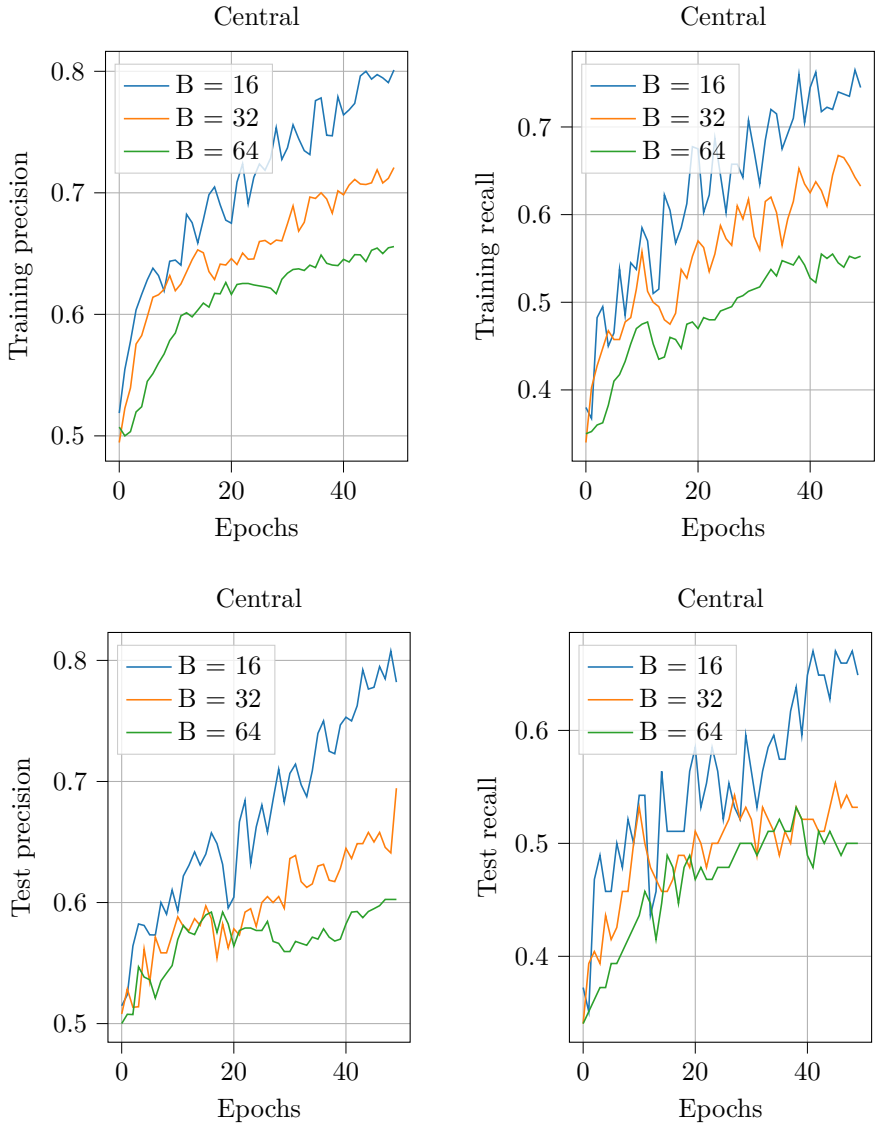


Figure B.1: Precision and recall during both training and testing of a centralized model with centralized rehabilitation data. The plots were made by varying training batch size.

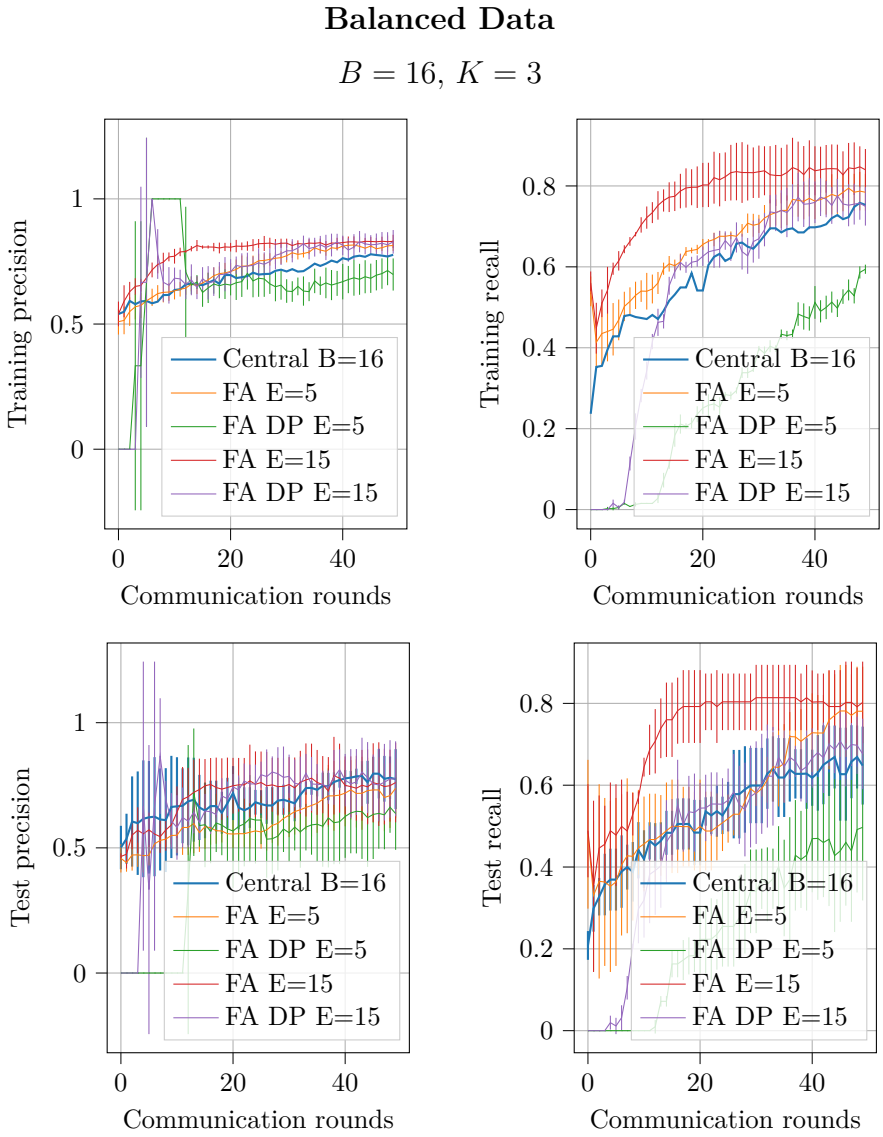


Figure B.2: Precision and recall during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different numbers of epochs. Training was done using the rehabilitation data. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

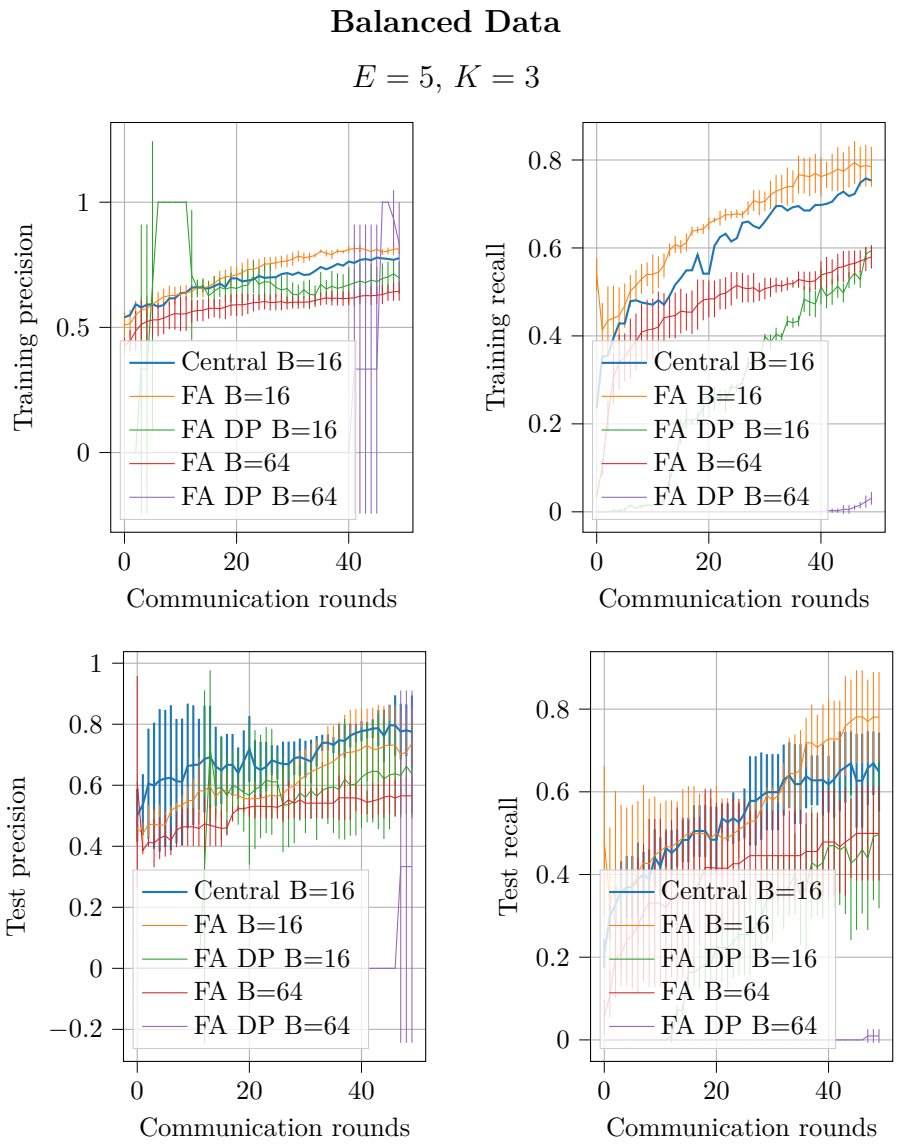


Figure B.3: Precision and recall during both training and testing of a global model trained using Federated Averaging with and without differential privacy for different numbers of epochs. Training was done using the rehabilitation data. The plots show the mean accuracies across all clients where the vertical lines show standard deviations. The data was balanced and not label skewed. Epochs size was set to $E = 5$ and number of clients $K = 3$.

The precision and recall for various batch sizes is seen in Figure B.5. We see here that the test precision and recall are lower for the majority of clients except for a few clients with a federated model where $B = 16$ (yellow).

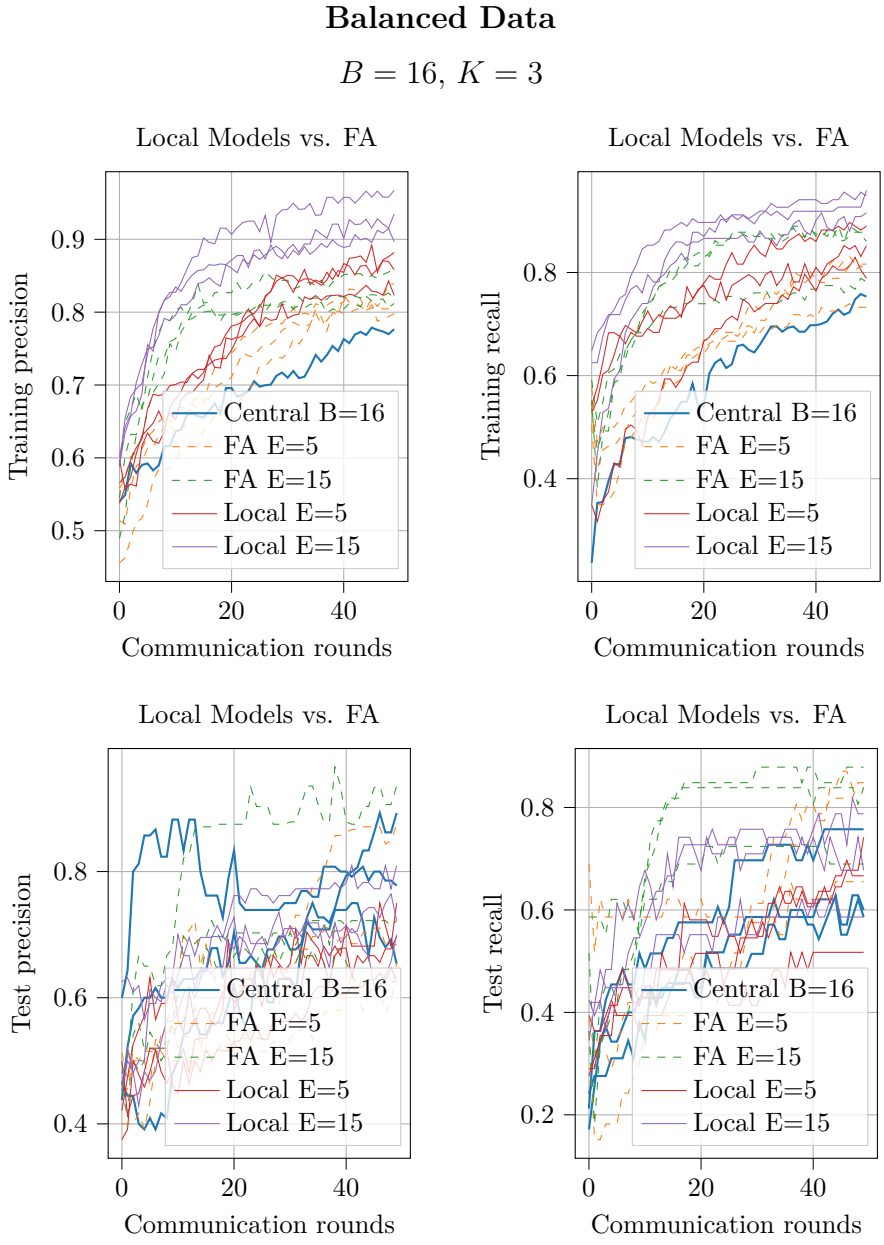


Figure B.4: Precision and recall during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

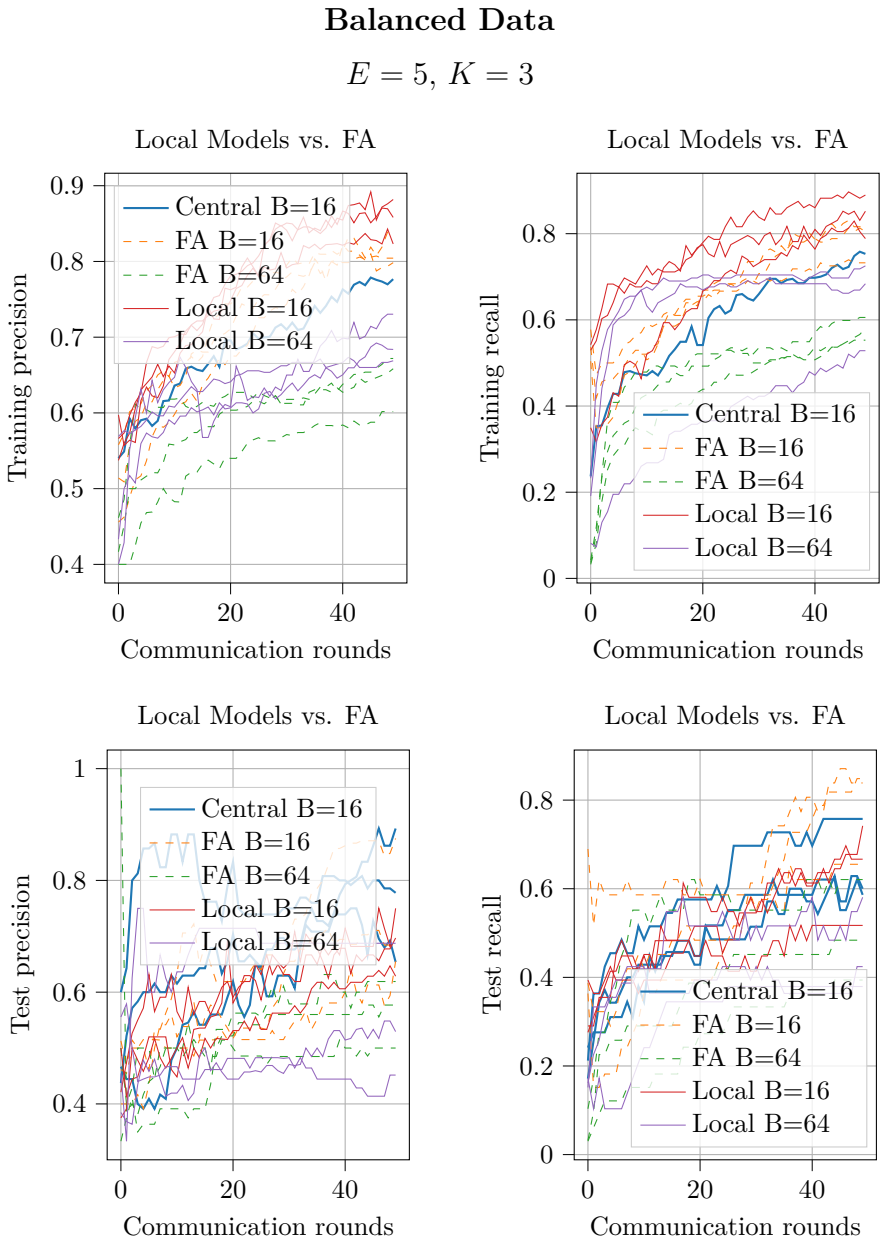


Figure B.5: Precision and recall during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

B.3.2 Unbalanced Data

In Figure B.6, we see that most of the local models generally perform well in test, though, some of the clients (red and purple) lie steadily on zero test precision and recall. We also see that some of the federated models lie steadily on the opposite end with perfect test precision (orange and green). This could indicate that more samples are needed for training and testing. Nonetheless, in a similar fashion as the corresponding MNIST experiment, we see that the federated algorithm generally is able to learn a model that also benefits the clients with less data. The same trends can be seen for various batch sizes Figure B.7.

B.3.3 Balanced Data with Label Distribution Skew

In Figure B.8 the test precision and test recall of the federated models generally reach high scores that surpass both the local models and the baseline. It can also be seen that the baseline test precision scores vary a lot between clients, but are rather consolidated for local and federated models. Finally, the test recall scores for the federated model where $E = 15$ (green) plateaus at roughly 80% to 90% after only about 10 communication rounds. In Figure B.9 we see that the precision and recall values during both training and testing drops significantly when increasing batch size to $B = 64$.

B.3.4 Unbalanced Data with Label Distribution Skew

In Figure B.10 we see again rather high variation in precision scores between clients of all models. In this case the worst performing local models reaches higher precision in fewer rounds compared to the worst performing federated models and baseline. The federated model where $E = 5$ (yellow) performs comparably to the central model with respect to precision. The federated models are generally more consolidated in the test recall scores than the local models, and reach an average of about 85%, whereas the centrally trained model is around 65%. In Figure B.11 we see that increasing the batch size to $B = 64$ decreases the test precision and test recall of the local models (purple). For the federated model, increasing batch size gives slightly decreased the recall scores with a bit more variation, however, the precision scores do not change much when comparing to $E = 15$ in the previous Figure B.10.

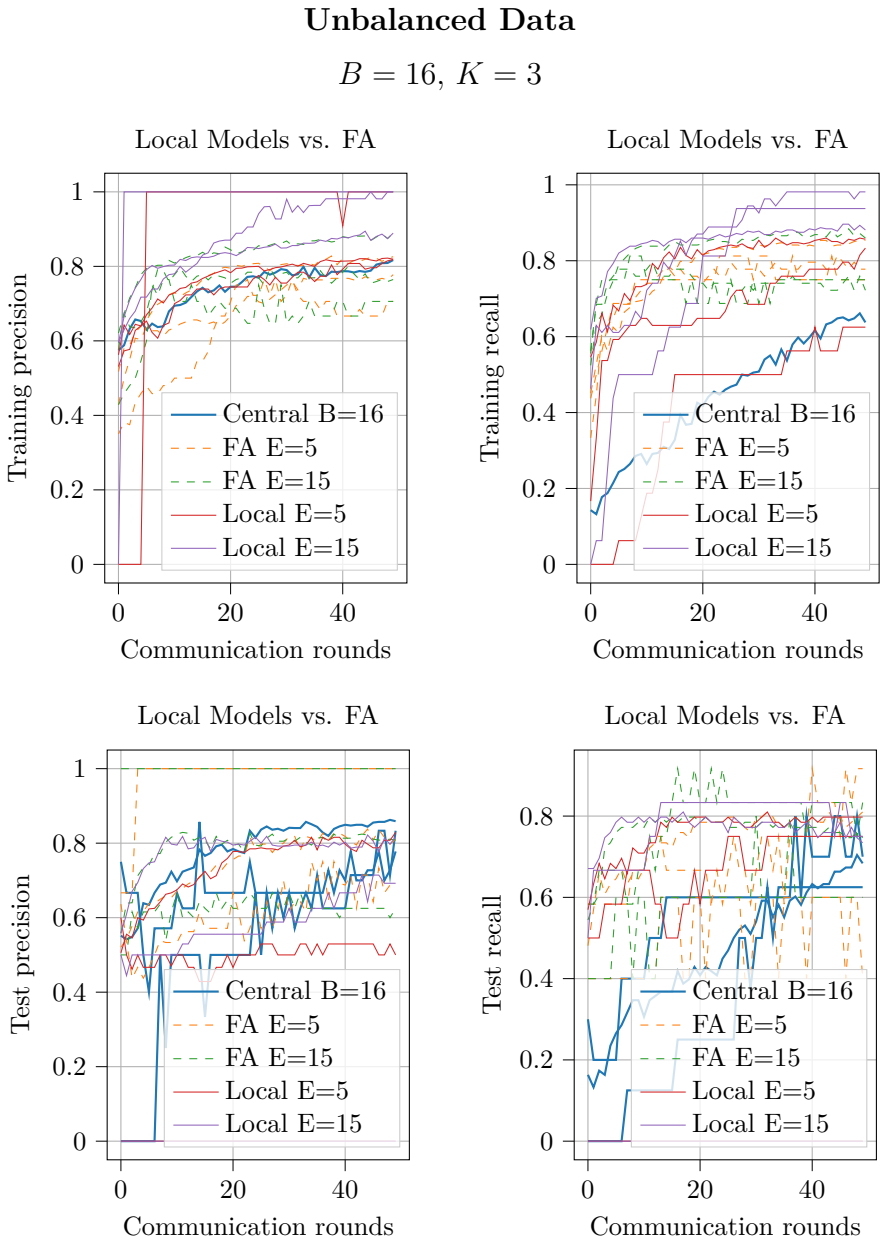


Figure B.6: Precision and recall during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and not label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

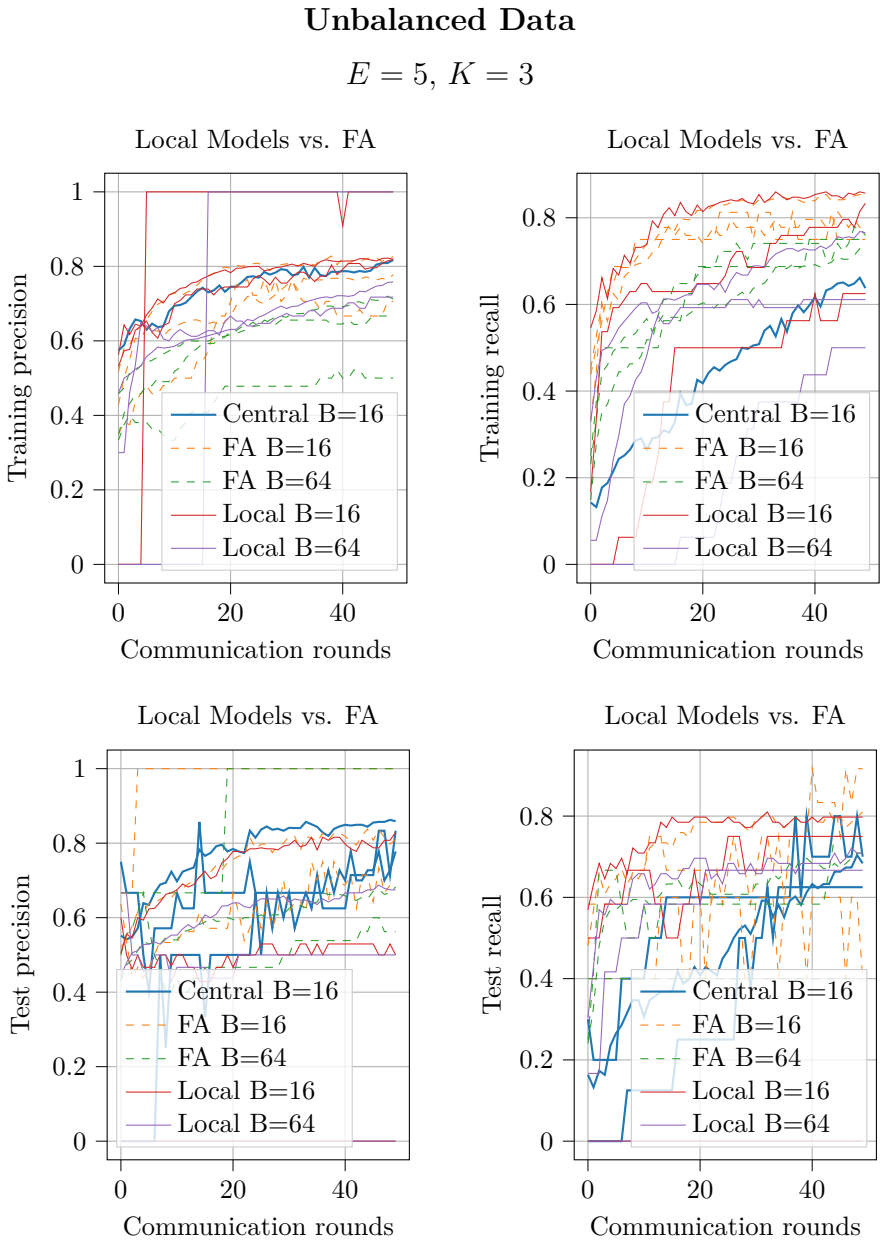


Figure B.7: Precision and recall during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and not label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

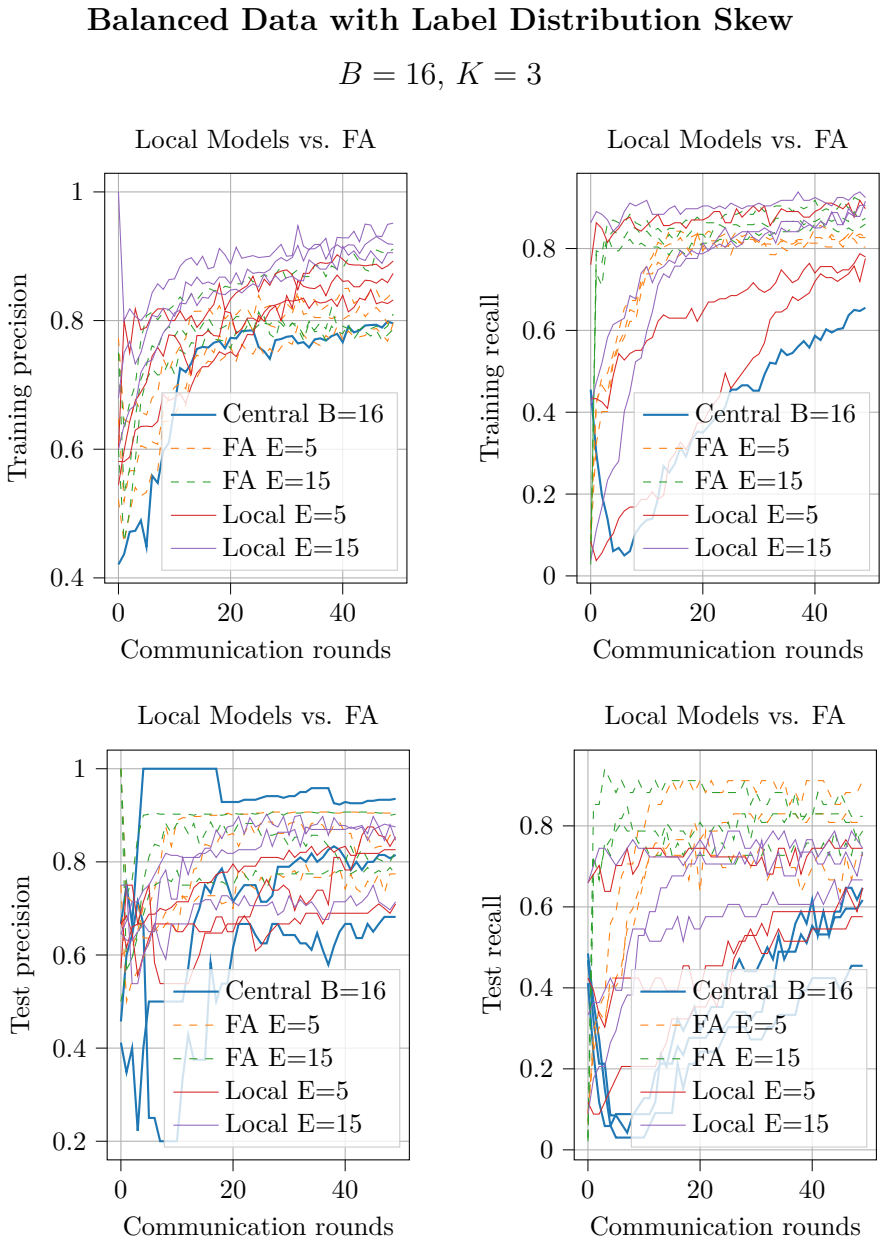


Figure B.8: Precision and recall during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

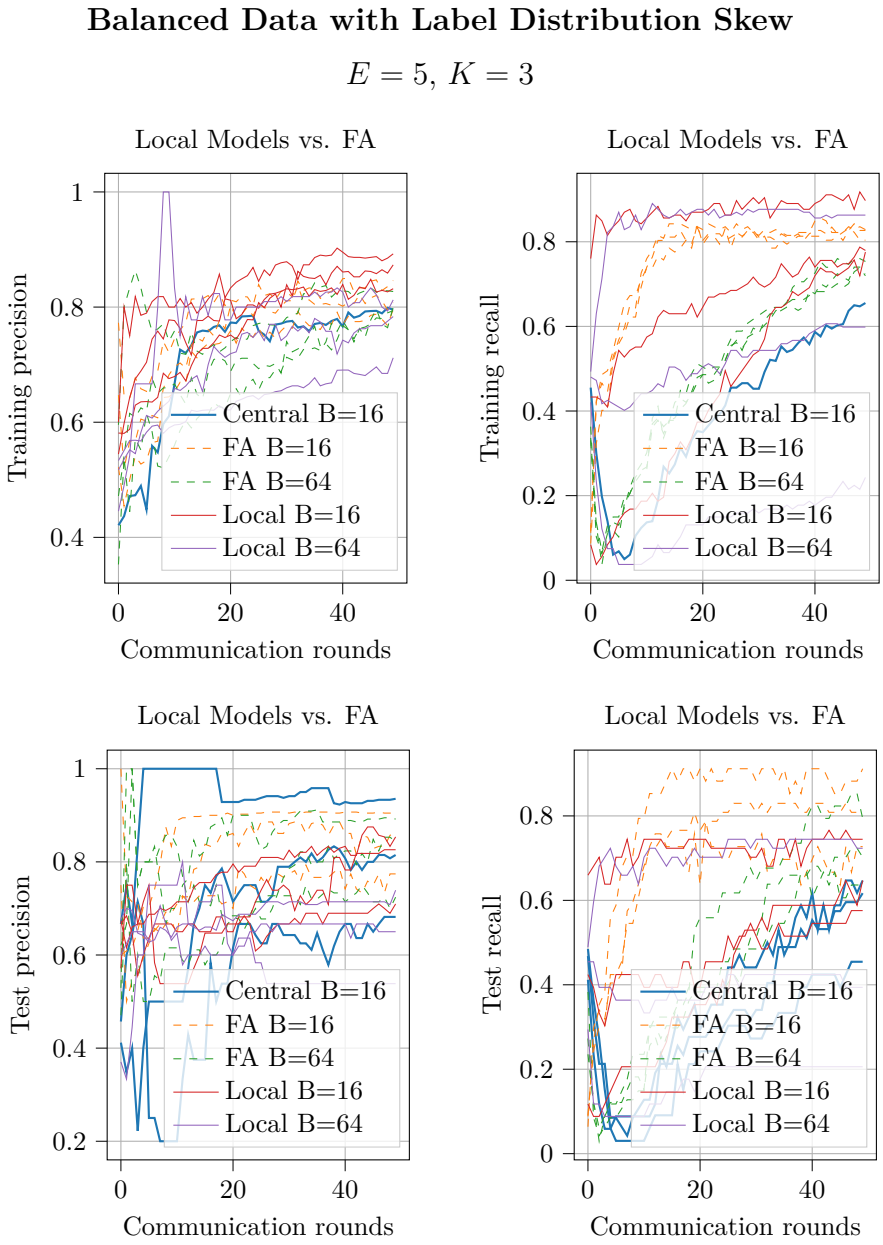


Figure B.9: Precision and recall during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was balanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

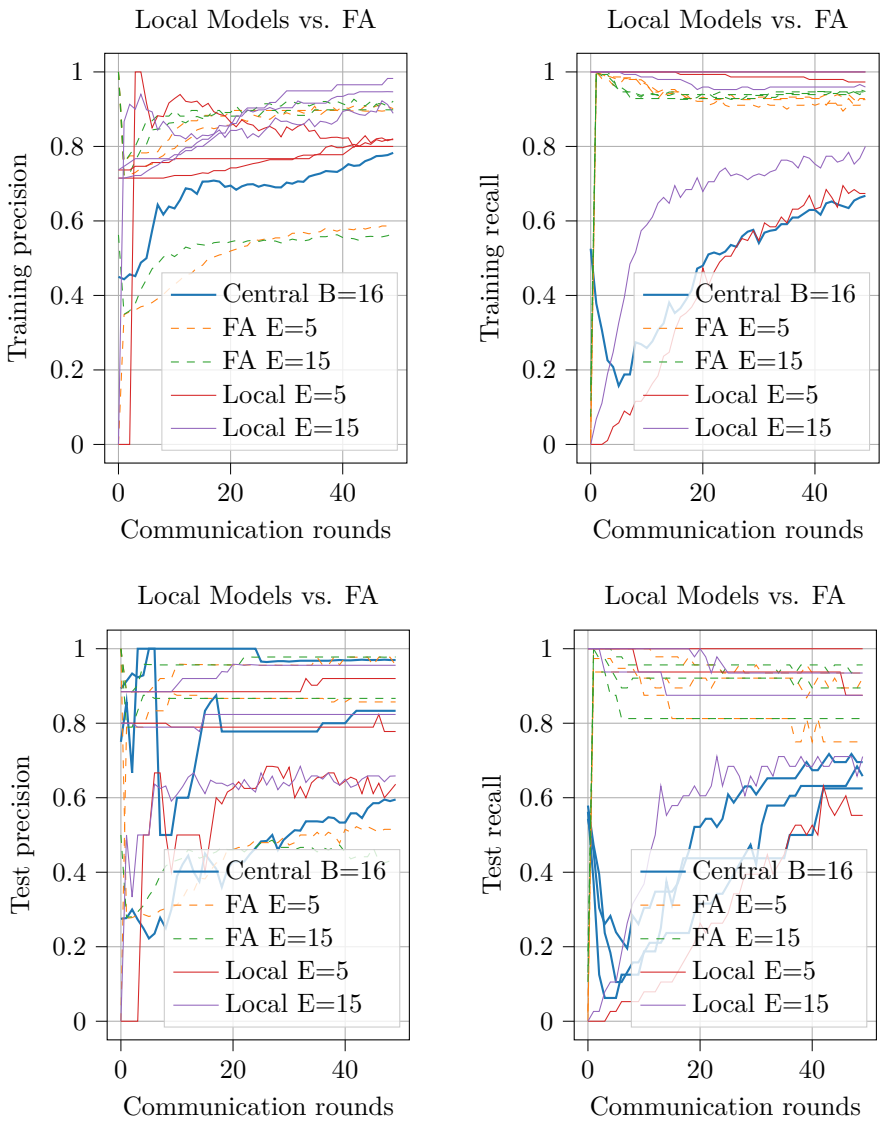


Figure B.10: Precision and recall during both training and testing of local models trained for different numbers of epochs. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and label skewed. Batch size was set to $B = 16$ and number of clients $K = 3$.

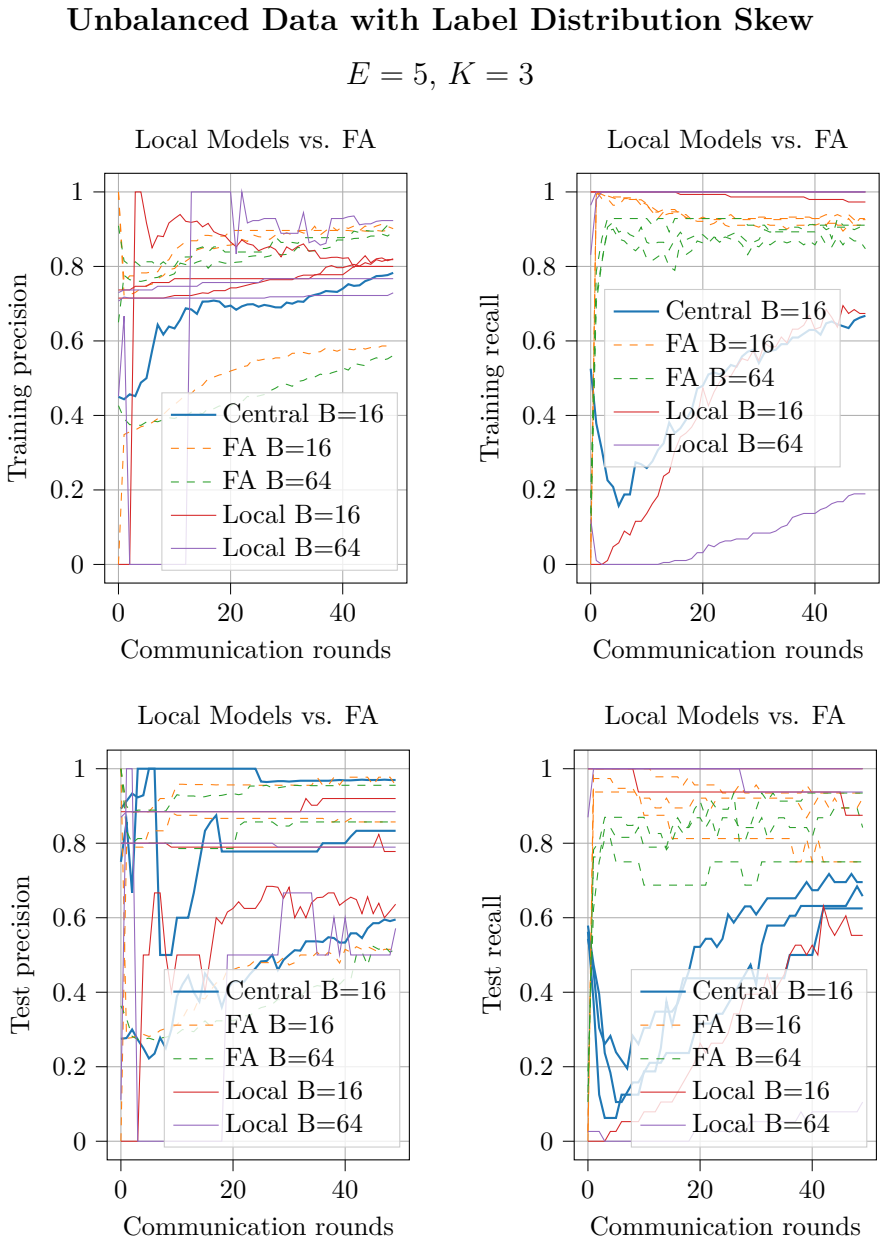


Figure B.11: Precision and recall during both training and testing of local models trained for different batch sizes. Each line corresponds to a single client. Training was done using the rehabilitation data. The data was unbalanced and label skewed. Epoch size was set to $E = 5$ and number of clients $K = 3$.

Abbreviations

AIR	AI-Rehabilitation
AUC	Area Under the Curve (of ROC)
DP	Differential Privacy
DPSGD	Differentially Private Stochastic Gradient Descent
ELU	Exponential Linear Unit
FA	Federated Averaging
FL	Federated Learning
FPR	False Positive Rate
GDPR	General Data Protection Regulation
IID	Independent and Identically Distributed
KL	Kommunernes Landsforening
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PFNM	Probabilistic Federated Neural Matching
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SMPC	Secure Multi-Party Computation
TPR	True Positive Rate

Nomenclature

Calculus

∇ Gradient.

∂ Partial derivative.

f' The single quote denotes the first derivative of f.

Experiment Parameters

B Batch size.

E Number of local epochs.

K Number of clients.

Linear Algebra

$[\,]$ Matrix or vector.

$\mathbf{W}$ Matrix denoted by capital bold letters.

$\mathbf{W}^T$ Transposition of matrix $\mathbf{W}$.

$\mathbf{y}$ Vector denoted by bold letter.

Number Sets

$\mathbb{F}_p$ Finite field of characteristic, or order, p.

$\mathbb{R}$ Real numbers.

Other Symbols

$\arg\max f(x)$ The value of x for which $f(x)$ is maximized.

Δ Change or difference.

δ In differential privacy this is the probability that information is leaked by accident.

δ_{ij} Kronecker delta function.

ϵ In differential privacy this is the privacy budget.

η	Learning rate.
$\hat{y}$	The accent circonflexe indicates an estimate of y.
$\leftarrow$	Assignment.
$\mathbf{x}$	Sample vector that is passed through a neural network.
$\mathbf{y}$	Target vector.
$\mathcal{P}_k$	The data partition for client k.

Probability

$\Pr(A)$	Probability of event A.
$\sim$	Is distributed as.

Bibliography

[1] Ministry of Social Affairs and the Interior, "Bekendtgørelse af lov om social service." https://www.retsinformation.dk/eli/lta/2020/1287, June 2020. Accessed: 2020-11-08.

[2] C. F. Smed, "Kunstig intelligens skal understøtte kommunale rehabiliteringsforløb." https://projekter.au.dk/fileadmin/projekter/air/Nyhedsbrev_maj_2020_AIR_projekt.pdf, May 2020. Accessed: 2020-11-08.

[3] C. F. Pedersen, "Projektbeskrivelse." https://projekter.au.dk/air/projektbeskrivelse, Sept. 2020. Description of the AIR project. Accessed: 2020-11-08.

[4] Benchmarking Unit of the Ministry of Social Affairs and the Interior, "Rehabilitering på ældreområdet efter §83a i serviceloven." https://simb.dk/media/37617/rehabilitering-paa-aeldreomraadet-efter-83a-i-serviceloven.pdf, Oct. 2019. Accessed: 2020-11-8.

[5] Digitaliseringsstyrelsen, "Digitaliseringspagt." https://digst.dk/media/19919/digitaliseringspagt-en-ny-retning-for-det-faellesoffentlige-samarbejde.pdf, Mar. 2019. Accessed: 2020-11-11.

[6] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, Jan. 2019.

[7] M. S. Gross and J. Robert C. Miller, "Federated learning: collaboration without compromise for health care research." https://www.statnews.com/2020/02/13/federated-learning-safer-collaboration-health-research, 2020. Accessed: 2020-10-24.

[8] The European Parliament and the Council of 27 April 2016, "Regulation (eu) 2016/679." https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679, May 2016. Accessed: 2020-11-10.

[9] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang,

L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1, 2021.

[10] H. B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data." `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, 2017. Accessed: 2020-10-24.

[11] Apple, "Designing for privacy." Apple WWDC, `https://developer.apple.com/videos/play/wwdc2019/708`, 2019. Accessed: 2020-11-14.

[12] S. R. Pfohl, A. M. Dai, and K. A. Heller, "Federated and differentially private learning for electronic health records," *arXiv e-prints*, vol. abs/1911.05861, 2019.

[13] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 267–284, USENIX Association, Aug. 2019.

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

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

[16] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *arXiv e-prints*, 2020, 1907.09693.

[17] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *arXiv e-prints*, 2020, 1911.06270.

[18] C. Tack, "Artificial intelligence and machine learning | applications in musculoskeletal physiotherapy," *Musculoskeletal Science and Practice*, vol. 39, pp. 164 – 169, 2019.

[19] W. O. Nijeweme-d'Hollosy, L. van Velsen, M. Poel, C. G. M. Groothuis-Oudshoorn, R. Soer, and H. Hermens, "Evaluation of three machine learning models for self-referral decision support on low back pain in primary care," *International Journal of Medical Informatics*, vol. 110, pp. 31 – 41, 2018.

[20] A. J. Viera and J. M. Garrett, "Understanding interobserver agreement: the kappa statistic," *Fam. Med.*, vol. 37, no. 5, pp. 360–363, 2005.

[21] L. Shahmoradi, Z. Liraki, M. Karami, B. alizadeh savareh, and M. Nosratabadi, "Development of decision support system to predict neurofeedback response in adhd: an artificial neural network approach," *Acta Informatica Medica*, vol. 27, pp. 186–191, Sept. 2019.

[22] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtarik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv e-prints*, 2016, 1610.02527.

[23] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, pp. 1–1, May 2020.

[24] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of Machine Learning Research* (A. Singh and J. Zhu, eds.), vol. 54, (Fort Lauderdale, FL, USA), pp. 1273–1282, PMLR, Apr. 2017.

[25] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, "Bayesian nonparametric federated learning of neural networks," in *Proceedings of Machine Learning Research* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97, (Long Beach, California, USA), pp. 7252–7261, PMLR, June 2019.

[26] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 1126–1135, PMLR, Aug. 2017.

[27] P. Sharma, F. E. Shamout, and D. A. Clifton, "Preserving Patient Privacy while Training a Predictive Model of In-hospital Mortality," *arXiv e-prints*, Dec. 2019, 1912.00354.

[28] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 111–125, IEEE, 2008.

[29] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 4768–4777, Curran Associates Inc., 2017.

[30] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Úlfar Erlingsson, "Scalable private learning with pate," in *International Conference on Learning Representations (ICLR)*, 2018.

[31] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1 – 8, 2018.

[32] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016.

[33] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *International Conference on Learning Representations (ICLR)*, 2018.

[34] A. Woodruff, "What is a neuron?." `https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron`, 2019. Accessed: 2020-09-18.

[35] Lumen Learning, "Neurons." `https://courses.lumenlearning.com/waymaker-psychology/chapter/cells-of-the-nervous-system/`, 2019. Accessed: 2020-09-20.

[36] QBI Laboratories, "Action potentials and synapses." `https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/action-potentials-and-synapses`, 2017. Accessed: 2020-09-20.

[37] A. Iosifidis, "Introduction to machine learning." Lecture notes from Optimization and Data Analytics at Department of Engineering, Aarhus University, 2018.

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

[39] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang, and S. Tandon, "Softmax regression." `http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression`, 2013. Accessed: 2020-10-7.

[40] E. Bendersky, "The softmax function and its derivative." `https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative`, 2016. Accessed: 2020-10-3.

[41] K. Janocha and W. Czarnecki, "On loss functions for deep neural networks in classification," *Schedae Informaticae*, vol. 25, Feb. 2017.

[42] E. K. P. Chong and S. H. Żak, *An introduction to optimization*. Wiley series in discrete mathematics and optimization, Hoboken, New Jersey: Wiley, fourth edition ed., 2013.

[43] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[45] Google, "Classification." `https://developers.google.com/machine-learning/crash-course/classification`, 2020. Accessed: 2020-10-24.

[46] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Inc., 2014.

[47] D. F. Aranha, "Fundamentals of computer security." University lecture at Department of Engineering, Aarhus University, 2019.

[48] Google, "How google anonymises data." `https://policies.google.com/technologies/anonymization`. Accessed: 2020-11-20.

[49] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," Tech. Rep. SRI-CSL-98-04, Computer Science Laboratory, SRI International, 1998.

[50] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, p. 211–407, Aug. 2014.

[51] A. Trask, "Secure and private AI." `https://classroom.udacity.com/courses/ud185`. Accessed: 2020-11-24.

[52] F. McSherry, "Privacy integrated queries," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, Association for Computing Machinery, Inc., June 2009.

[53] Y. Zhao, J. Zhao, M. Yang, T. Wang, N. Wang, L. Lyu, D. Niyato, and K.-Y. Lam, "Local differential privacy based federated learning for internet of things," *IEEE internet of things journal*, pp. 1–1, 2020.

[54] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits," in *Computer Security – ESORICS 2013* (J. Crampton, S. Jajodia, and K. Mayes, eds.), (Berlin, Heidelberg), pp. 1–18, Springer Berlin Heidelberg, 2013.

[55] V. Pastro, *Zero-Knowledge Protocols and Multiparty Computation*. PhD thesis, Department of Computer Science, Aarhus University, July 2013.

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

[57] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv e-prints*, 2019, 1802.03888.

[58] Y. LeCun, C. Cortes, and C. J. Burges, "MNIST handwritten digit database." `http://yann.lecun.com/exdb/mnist/`. Accessed: 2020-10-24.

[59] Google, "Embeddings." `https://developers.google.com/machine-learning/crash-course/embeddings`, 2020. Accessed: 2021-1-4.