UPPSALA
UNIVERSITET

UPTEC F 21024
Examensarbete 30 hp
Juni 2021

Decentralized machine learning on massive heterogeneous datasets

A thesis about vertical federated learning

Oskar Lundberg

Civilingenjörsprogrammet i teknisk fysik

UPPSALA
UNIVERSITET

Decentralized machine learning on massive heterogeneous datasets

Oskar Lundberg

Abstract

The need for a method to create a collaborative machine learning model which can utilize data from different clients, each with privacy constraints, has recently emerged. This is due to privacy restrictions, such as General Data Protection Regulation, together with the fact that machine learning models in general needs large size data to perform well. Google introduced federated learning in 2016 with the aim to address this problem. Federated learning can further be divided into horizontal and vertical federated learning, depending on how the data is structured at the different clients. Vertical federated learning is applicable when many different features is obtained on distributed computation nodes, where they can not be shared in between. The aim of this thesis is to identify the current state of the art methods in vertical federated learning, implement the most interesting ones and compare the results in order to draw conclusions of the benefits and drawbacks of the different methods. From the results of the experiments, a method called FedBCD shows very promising results where it achieves massive improvements in the number of communication rounds needed for convergence, at the cost of more computations at the clients. A comparison between synchronous and asynchronous approaches shows slightly better results for the synchronous approach in scenarios with no delay. Delay refers to slower performance in one of the workers, either due to lower computational resources or due to communication issues. In scenarios where an artificial delay is implemented, the asynchronous approach shows superior results due to its ability to continue training in the case of delays in one or several of the clients.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Selim Ickin Ämnesgranskare: Thomas Schön

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

Inom teknologisektorn har maskininlärning och dess förmåga att utnyttja data för att lära sig modeller utvecklats enormt under de senaste årtiondena. Dessa modeller, som används för att kunna förutspå utkomsten av specifika problem med hjälp av ny data, kräver generellt mycket data för att kunna träna modellen. Dataskyddsförordningen (GDPR) och andra sorts regler om hur data ska skyddas har skapat isolerad data, data som alltså inte får lämna specifika klienter. Denna rapport handlar om vertical federated learning, en metod som kan utnyttja skyddad data från flera olika klienter för att skapa en gemensam maskininlärnings modell, utan att bryta dessa skyddsförordningar. Vertical federated learning är en relativt ny metod och denna rapport behandlar behovet av att sammanfatta dom existerande state of the art (SOTA) metoderna som finns inom vertical federated learning. Några av dessa metoder implementeras och testas för att kunna dra slutsatser om varje enskild metods för- och nackdelar. Från resultaten så visar metoden FedBCD väldigt lovande resultat genom att minska antalet kommunikationsrundor som krävs för att modellen ska konvergera, detta genom att utföra mer beräkningar hos varje enskild klient. Förutom detta, så utförs experiment för att jämföra resultaten då kommunikationen sker synkront eller asynkront. Resultaten visar att en synkron kommunikation är marginellt bättre under ideala förhållanden. Sedan utförs två olika experiment där klienterna utsätts för fördröjningar, för att spegla en verklig implementation av vertical federated learning. Från dessa experiment visar en asynkron kommunikation stora fördelar, då den har förmågan att fortsätta träna även om en eller flera klienter upplever fördröjningar.

Contents

Chapter 1

Introduction

This thesis addresses a vertical federated learning problem, where the objective of this thesis is to investigate the different state of the art methods in vertical federated learning to draw conclusions of each methods benefits and drawbacks. For the problem formulation we assume $\mathcal{K}$ clients and one master that collaboratively learns a model from the data. Each client has access to a subset of the features of the data, which is what defines a vertical federated learning problem. The goal is to train a collaborative model without violating any privacy constraints which exists in a vertical federated learning system. This thesis is performed on the behalf of the "Data Science & Automation R&D" team at Ericsson in Kista. In the different areas that Ericsson are involved in, different vertical federated learning scenarios naturally occurs. Therefore, Ericsson has an interest in mapping and evaluating the different methods that exists in vertical federated learning.

Machine learning and artificial intelligence has progressed massively during the last couple of decades. Machine learning utilizes data to learn a model that can predict the outcome of a problem when exposed to new data and its application area is enormous, in areas such as computer vision, medicine, speech recognition and natural language processing amongst others [1]. As machine learning evolves and is implemented to solve more complex problems, large size data is needed to solve these problems. As stated by Norvig et al. *"simple models and a lot of data trump more elaborate models based on less data"* [2]. New data privacy restrictions, such as General Data Protection Regulation (GDPR), has created instances of "data islands", i.e. isolated data located at different clients [3]. These "data islands", together with the fact that machine learning models generally are data hungry, i.e. they need a lot of data to perform well, has created the need for a method to create a collaborative machine learning model which can utilize data from several different clients without violating privacy constraints or revealing any of the sensitive data. In 2016, Google introduced federated learning [4] with the aim to solve this problem. The idea is to train a collaborative model where the data is distributed across several devices where the training is performed in such a way that the individual client's raw data is never revealed. Google later also then applied this to improve the auto completion of typing words and sentences on the keyboards of cellphones [5], thus utilizing data from many different phones, each with privacy constraints, to create a combined model. As Federated learning has evolved, the interest and applications of it has grown and now also Apple utilizes Federated learning for their keyboards as well as the "Hey Siri" vocal classifier [6]. Federated learning can further be divided into horizontal and vertical federated learning, depending on how the data is structured at the clients, where this thesis focuses on the vertical case.

Federated learning is a recently emerged and growing field of research and since it was first introduced, its rapid progress has been astonishing as illustrated in Figure 1.1.

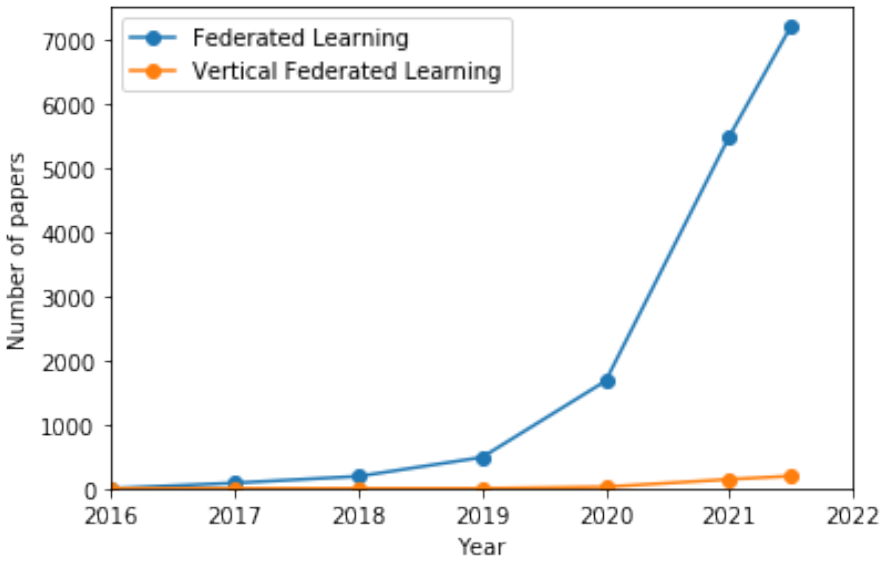


Figure 1.1: Illustration of the number of papers released about federated learning vs vertical federated learning over time. The figure is created by the number of hits on google scholar for "federated learning" and "vertical federated learning".

New State of the art (SOTA) methods and techniques are frequently introduced in papers of federated learning, but the majority of these papers covers horizontal federated learning, often also only referred to as "federated learning", and not the vertical variant, as seen in Figure 1.1. This has created a need to explore the lesser researched of the two to explore which SOTA methods exists in vertical federated learning and how these methods compare to each other.

Since federated learning is a relatively new method, several large survey papers exists ([3], [7], [8], [9], [10]). The papers try to summarize the work and methods established so far in this subject and gives a a good overview and introduction on federated learning in general. Again, since most of the work done so far in federated learning mostly covers the horizontal case, so does these survey papers.

The authors of Advances and Open Problems in federated learning [8] proposes the following definition for federated learning:

"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".

The goal of this thesis is to investigate which vertical federated learning SOTA techniques exists, implement the most interesting ones and then compare the results of the methods in order to draw conclusions of the benefits and drawbacks of the different methods.

Chapter 2

Neural networks

Neural networks are a popular and powerful machine learning model that tries to emulate the learning mechanism of biological organisms. They are flexible and can model and reproduce complex nonlinear relationships. Since all the vertical federated learning methods covered in this thesis utilizes neural networks as the underlying machine learning model, this chapter will describe the basics of neural networks.

Neural networks are built by layers, where each layer consists of a number of nodes that are called neurons. The number of layers to use and the number of neurons in each layer is a design choice which can be optimized depending on the problem and depending on the desired complexity of the network, since more layers and neurons results in more parameters to optimize during training. The general structure of a neural network, illustrated in Figure 2.1, consists of an input layer, hidden layers and an output layer. The input layer consists of as many neurons as there are features in the input data. The neural network then can have one or several layers, called hidden layers. The neural network's complexity increases with the number of hidden layers used and if it utilizes many hidden layers it can be refereed to as a "deep neural network". Finally there is the output layer, which then acts as the prediction of the network. The ouput layer's objective is to output a number that serves as the prediction for the problem. For example, for a binary classification problem, the output can be a probability (a number between 0 and 1) and can be interpreted as how certain the model is of its prediction. For a regression problem the output is the predicted value of the quality of interest.

To convey the information forward through the network, each neuron is linked to the neurons in the next layer with adaptive weights, which then affects how much influence one neuron has on another. The value of a neuron in the next layer is calculated by the weighted sum of all the values of the neurons connected to the neuron plus a bias. This value then gets put through an activation function. The weights and biases are the neural networks parameters which changes during training, with the objective to find the values that can best model the problem at hand. The activation functions are what makes neural networks able to predict complex relationships and the most commonly used are "Linear", "Sigmoid", "Sign", "Tanh" and "Rectified Linear Unit (ReLu)" and their respective equation is shown in Table 2.1.

Linear	Sigmoid	Tanh	ReLu	Sign
$f(x) = x$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f(x) = max(0, x)$	$f(x) = sign(x)$

Table 2.1: The most common activation functions for neural networks.

To train a neural network it goes through two stages, forward- and backpropagation. Forwardpropagation is the process of feeding data into the network and let it propagate through the entire network to the output layer. This is done by starting from the input data, using the current set of parameters, calculate the values of the neurons in the next layer by taking the sum of all the neurons connected to it, multiplied with their respective weights plus the bias. This value then gets put through the chosen activation function to get the value for the neuron. This process is then repeated until the output layer and thus a prediction is calculated. At the output layer the ground truth, i.e. the label of the problem,

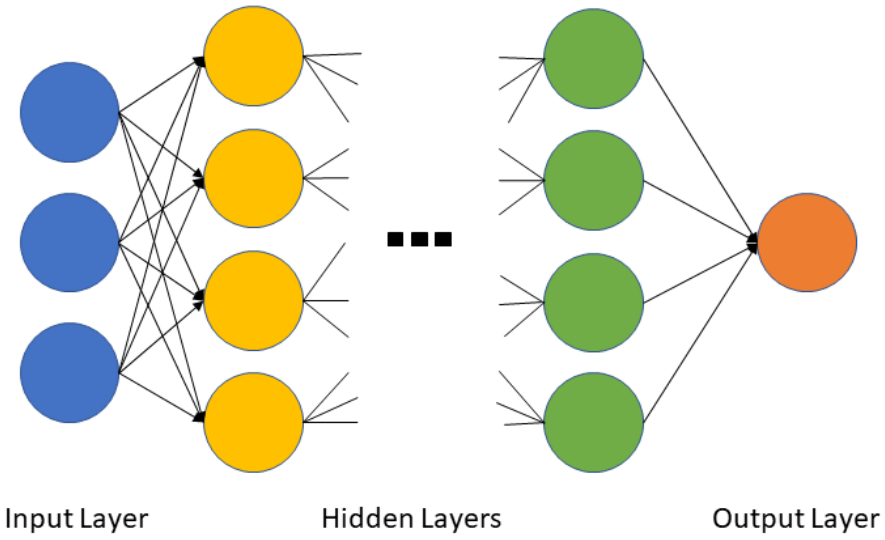


Figure 2.1: General structure of a neural network.

comes in to be used in the loss function to calculate how far off the model's prediction was. The process of forwardpropagation is also performed on the complete trained network to make predictions on new data.

Backpropagation is the process of calculating how much each parameter affected the prediction and to change the parameters to improve future predictions. This is done by calculating partial derivatives for each parameter starting from the loss function at the end of the neural network all the way until the first layer of the model by utilizing the chain rule from differential calculus. By calculating how much the error of the loss function depends on the neuron in the output layer, we can then utilize the chain rule to calculate the error of the neurons in the previous layer based on the error of the neurons in the current layer. This is repeated until the error of the neurons in the input layer is calculated. Based on these errors of each neuron in each layer, the partial derivative of how much each weight and bias affected this error can be calculated. Finally the parameters can be updated by multiplying each parameters partial derivative with the learning rate and subtracting that value from the current value of each respective parameter. The learning rate is a hyper-parameter and a design choice, which determines how much to change the model's parameters, the weights and the biases, by multiplying the learning rate with the partial derivatives derived from the backpropagation. The loss function computes the error in the model's prediction based on the prediction of the neural network and the actual labels, the ground truth of the problem. During training, the goal is to minimize the loss function [11].

Common problems in training a neural network can be overfitting, i.e. fitting a model to a particular training data set which then does not guarantee that the model will perform well on test data, which is unseen data that is never used during training. This is connected to the complexity of the network and the amount of training data available. In general, the generalization power is reduced as a more complex model is utilized but increased as the number of training instances are increased. Complex problems often requires a complex model to be solved, where poor performance can be the results of lack of available data. This is also a reason for why federated learning exists, a way to be able to utilize more data, under privacy constraint, to be able to solve more complex problems.

Chapter 3

Federated learning systems

Since federated learning is a relatively new method, different papers have taken on the task of defining what a federated learning system consists of. Li et al. [7] defines a federated learning system by dividing it into subgroups (see Figure 3.1) where each subgroup and thus the entirety of a federated learning system will be described in this chapter.

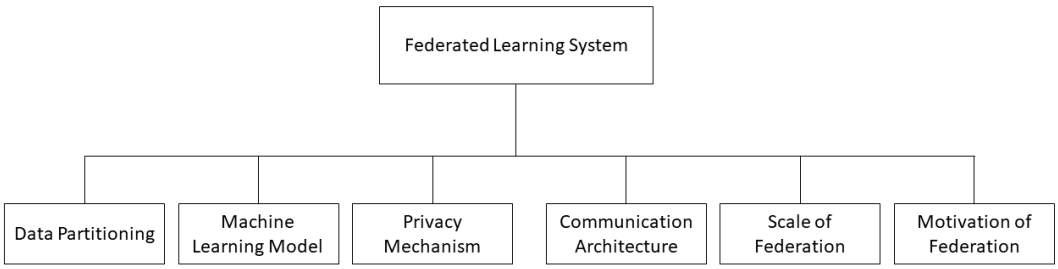


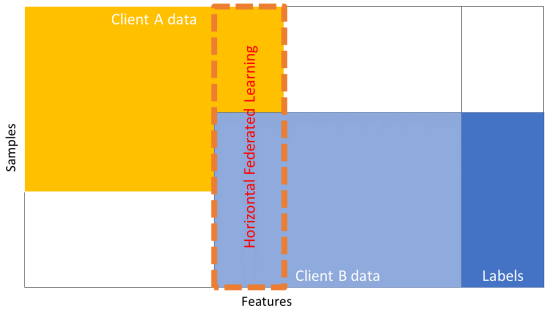
Figure 3.1: A federated learning system divided into its subgroups.

3.1 Data partitioning

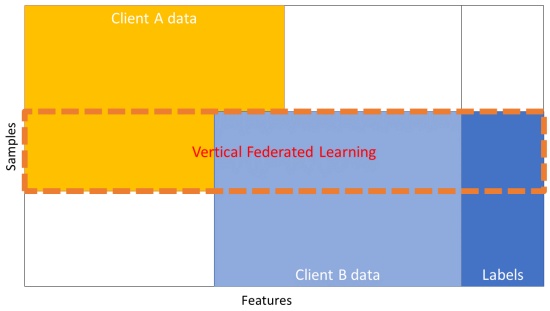
Depending on the data distributions characteristics, federated learning can be divided into horizontal federated learning (hFL) and vertical federated learning (vFL). Horizontal federated learning, also called data-partitioned, refers to when the data located at the different clients is partitioned by samples, meaning that the clients' data share the feature space but are different in the sample space. For vertical federated learning, also called feature-partitioned, the data at the different clients has the same sample space but differs in features. This thesis will focus on the vertical federated learning setting.

The difference in the data structure, thus resulting in either a horizontal or vertical federated learning problem, also affects the structure of the federated learning system and how it works. In general, for the horizontal case, each client trains a model, with the same machine learning model architecture across all clients, using their own data samples with their own labels. After a certain number of rounds a server collects, in a secure manner, and aggregates all the clients model parameters to get a global model which is an average of all local models. Then in a secure manner, the server sends the updated model back to the clients and the process repeats.

For the vertical case, each client here instead trains with the same samples but different features. Since they usually do not have access to the labels they instead send an intermediate result to a server or the client who has access to the label who then combines (in some manner) the intermediate results from all the clients to create one prediction. This client, often referred to as master, then can calculate the loss and gradients and then sends each client their respective gradient. A more detailed description of vertical federated learning and its training procedure will be provided in Chapter 4.



(a) Horizontally partitioned data.



(b) Vertically partitioned data.

Figure 3.2: Illustration of horizontally and vertically partitioned data.

Due to the difference in data structure which results in a difference how a horizontal and vertical federated learning system works, they each have their own algorithms and methods to learn a combined model. As stated, this thesis will focus on vertical federated learning and its state of the art methods.

3.2 Machine learning models

In a federated learning system, one has to decide which underlying machine learning model to use. Both in the horizontal and vertical federated learning case, many different machine learning models are applicable. Some vertical federated learning methods are based on specific machine learning models, such as Secureboost: A lossless federated learning framework [12] which utilizes gradient boosting decision trees as the machine learning model. However, most of the federated learning methods are based on stochastic gradient descent, thus making machine learning models that utilizes stochastic gradient descent potential choices, such as support vector machines, logistic regression and neural networks. Most commonly used is neural networks, due to its high performance in many different machine learning tasks. Since most state of the art methods for vertical federated learning is based on stochastic gradient descent, this thesis will use neural networks as the underlying machine learning model in the federated learning system.

3.3 Privacy mechanism

Federated learning's training methodology focuses on privacy, since its use case is suitable when the data is distributed at different clients under privacy restrictions. Because of this, the privacy preserving mechanisms in a federated learning system is important to ensure that no data or information is leaked as well as making sure that the system is not vulnerable to attacks. In the vertical case, the sending of intermediate results and gradients between the clients and the master raises the question whether any information about the data at the clients can be extracted from these messages and if the system is vulnerable to attacks by tampering with these values. Different cryptographic methods like homomorphic encryption and secure multi-party computation are commonly used methods to ensure the privacy [13], [14]. In general they work by the sender encrypts its message before sending which subsequently is decrypted at the receiver. Another method commonly used is differential privacy, where noise is added to the clients results to further reduce the chances of a malicious party inferring any information. Usually by applying these above methods, the privacy in a federated learning system can be ensured but on the other hand it also inevitable increases the computational and the communication overhead as compared to a centralized approach where no encryption is needed [7].

3.4 Communication architecture

Another important distinction in federated learning systems is whether the system is centralized or decentralized. With centralized systems a server, often called master, is designated to be the coordinator of the training process. The communication between the clients and the master can then be either

synchronous or asynchronous. One example of a centralized design is Google's implementation to improve their keyboard predictions [5]. For the decentralized case, no such server exists and instead the clients themselves collaborate to perform the training. A centralized design creates further security concerns if the server cannot completely be trusted. Similar challenges also exists in the fully decentralized setting, where the clients need to trust each other. Therefore, the possibility to use blockchain [15] as communication architecture in a decentralized federated learning system is a recent approach that can be considered.

For the vertical scenario, a centralized design is most commonly used since usually only one party has access to the labels. This party then acts as a server or master for the system and is responsible to calculate the loss and send each other clients their respective gradients. A centralized design is what will be used in this thesis.

3.5 Scale of federation

Federated learning systems are usually divided further into either cross-silo or cross-device, depending on their scale. Cross-silo refers to the case when the number of involved clients are relatively low. The clients can be organizations or data centers, and usually has access to large amount of data as well as computational power. This setting is more common in the vertical case, when a handful of businesses or institutions want to collaborate and learn a joint model. Cross-device on the other hand refers to when the number of devices are relatively large and is more common in the horizontal case. Here each client usually has a relatively small amounts of data and less computational power then in the cross-silo setting. An example of this can be Google's implementation to train a model from many mobile devices, where each device has different samples and the data is under privacy constraints, but has the same features to train a joint model to improve the keyboard predictions [5].

3.6 Motivation of the federation

In a real application scenario, a party of the federated learning system needs some kind of motivation to be a part of the federation. This motivation can be in form of incentives or regulations. Incentives can for example be that the involved client has some kind of benefit or use case from the resulting model, and thus benefits from being part of the federated system. An example of regulations as motivation can be a federated learning system where the clients are within the same company but in different departments. Here the company can then have regulations such that they can utilize the data from the different clients.

Chapter 4

Vertical federated learning

This chapter will further explain how a vertical federated learning system works in detail and its training procedure as well as the different areas of improvement and the current state of the art methods in vertical federated learning.

As a comparison, in the horizontal setting each client can train their respective model simultaneously since each client has its own set of samples and labels. The server's responsibility is to synchronize after a certain number of rounds, where then each client sends their model parameters to the server who then aggregates them and sends the new updated parameters back to the clients. In the vertical setting, the master node, which is the node who has access to the labels, is here instead in charge to gather each clients intermediate results, combining them in some manner before calculating the loss and sending the gradients to each clients so that they can update their own model.

A general vertical federated learning structure is illustrated in Figure 4.1.

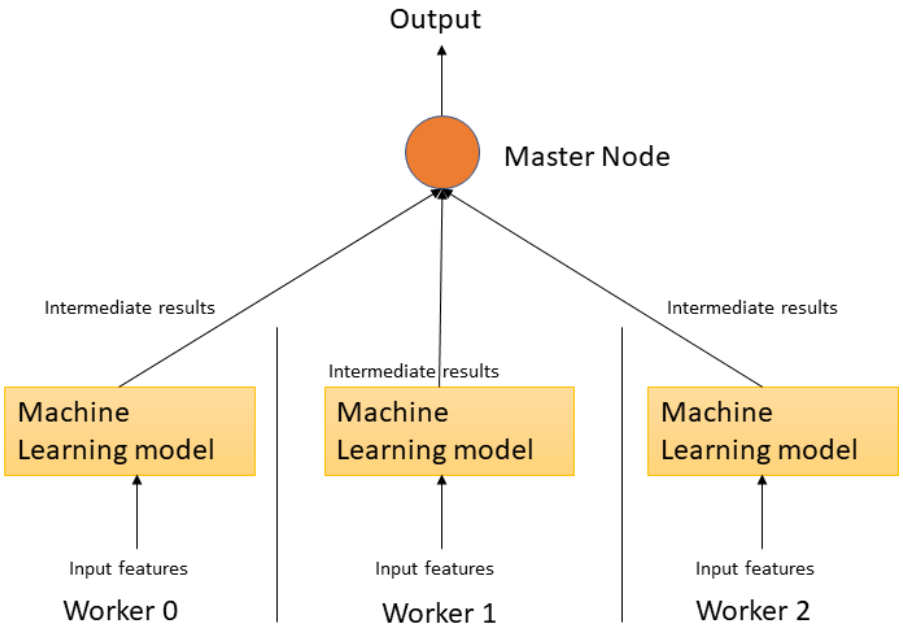


Figure 4.1: A general structure for vertical federated learning where each different worker has different input features.

Here the general idea is that each client is in charge of its own machine learning model. Each client's machine learning model can be differently structured depending on how many features it has access to. Each client uses its own feature space as inputs and the individual client model's output is that client's intermediate result. All clients' intermediate results are then sent, in a secure manner, to the master node, which depending on the problem formulation may or may not have access to data as well. The

master node then aggregates the intermediate results by concatenation and uses them as input to its own machine learning model to calculate the prediction. Since the master has access to the labels, it then can calculate the loss and gradients which it then sends to the respective client who then can update their model.

Also, since in vertical federated learning each client have different samples, an important first step in all vFL methods is entity alignment techniques. The goal is to in a secure way find and align all the samples that overlap for all clients so that the training can proceed [3]. In this thesis, we have assumed that the samples are already aligned and no entity alignment technique is needed.

4.1 SplitNN

One particular vertical federated learning structure and the structure that will be used for this thesis, is split neural network (SplitNN), where neural networks are used as the machine learning model and is split in such a manner so that each client has its own part of the neural network [16]. The clients use their own features space as input to its neural networks and then calculates an intermediate result at the so called cut layer, which can consist of several neurons. The intermediate result at the cut layer of each client is then sent to a server or master who has access to the labels. The master then uses all clients' intermediate results as input to its own neural network where the final output is the prediction for the problem. The master can then calculate the loss and start the backpropagation up until its cut layer, where it sends each client their respective gradient. The clients can then continue with the backpropagation using the gradient recieved from the master and finally every involved party can update their model.

The training procedure for SplitNN goes as follows:

1. The clients start forward propagation up until their cut layer, on the batch of data.
2. Each client then encrypts and sends their intermediate result to the master.
3. The master decrypts the messages and continues the forward propagation on its own neural network using all clients' intermediate results as input.
4. The master calculates the loss and starts backpropagation until its cut layer.
5. The master encrypts and sends each client their respective gradients as well as an index for the next batch of data and updates its own model.
6. The clients decrypts the message from the master and continues with the backpropagation with the gradient recieved from the master and finally updates their respective models.
7. This procedure is repeated until convergence.

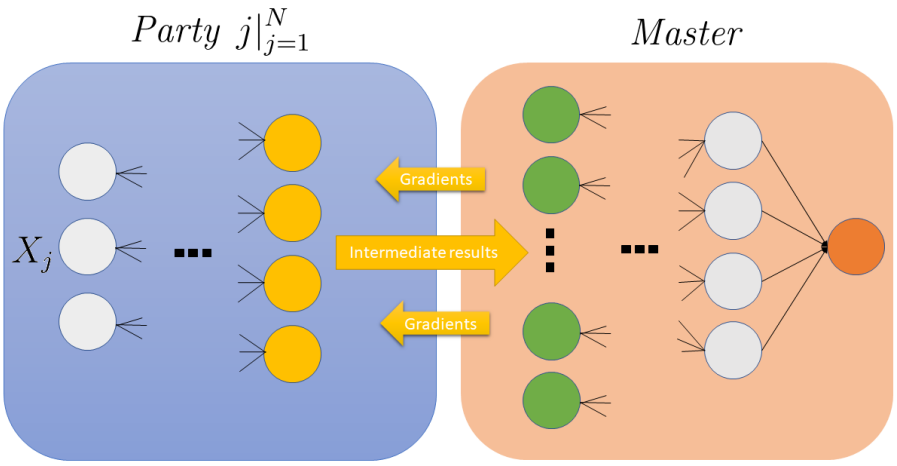


Figure 4.2: A general structure for SplitNN.

A SplitNN is illustrated in Figure 4.2 where here it is assumed that the master only has access to the labels, but not any input attributes. Due to the fact that SplitNN basically is one large neural network, with different parts distributed at different clients, it is very flexible and can take different structures. For example, in a setting where one party has access to both data and labels, one can utilize U-shaped SplitNN [17], where one can let every party send their intermediate result to a designated server, without access to either data or label. The server, who has a large neural network, can calculate and send its intermediate result back to the client who has access to the label. This client then has a small neural network to calculate the final prediction and loss, and the backpropagation is then started at this client. Most commonly is the assumption of a master who only has access to the labels and not the input attributes, which is the setting of this thesis.

4.2 Areas of potential improvements in vertical federated learning

This section covers the different areas for potential research where improvements in these areas would result in a better vertical federated learning system. These are the areas which new state of the art methods generally try to improve in, which then would result in a better performing method. In general these areas try to combat the potential bottlenecks in the federated learning systems, which often can be either within communication or computations [8].

4.2.1 Privacy

The privacy of a federated learning system is of vital concern, since the main goal for federated learning is to train a collaborate model without violating privacy constraints. Often, the assumption is made that all involved parties are "honest but curious" or "semi-honest" [18]. This means that one make the assumption that each involved client will follow the systems protocol and not cheat or do anything malicious. Nevertheless, they are curios about the data from other clients and can use the information available to them to try to infer the data of others. In this case, the privacy method from Section 3.3 is sufficient to ensure the privacy of the data and the federated learning system. If some client is not honest, which can happen in a real application scenario, the system might be vulnerable to attacks. So the question arises whether or not any information can be extracted from the encrypted intermediate results and gradients if one client has bad intents. Or can the performance of the model be tampered with if one client deliberately is sending bad data? Therefore extensive research is needed on the subject to further ensure privacy of the system and potentially reduce the computational overhead needed to ensure the privacy and is a current research area in federated learning [18], [19], [20]. That being said, this thesis will not focus on privacy concerns but instead assume that all clients involved are "honest but curious".

4.2.2 Communication

Since information has to be transmitted between the involved parties each round, the communication can often act as the bottleneck of the system [21]. Thus, methods that reduces the amount of communication is of interest. This can either be done by algorithms that can reduce the communication frequency or by reducing the size of each communication through compression. To reduce the communication frequency, one question can be whether or not the new information is needed or if a client can utilize the last known information instead. Again, since horizontal federated learning is the more researched case, more approaches that try to reduce the communication overhead exists for that [22] and further work is needed for potential improvement in vertical federated learning.

4.2.3 Model accuracy

The goal of any machine learning model is to achieve good prediction result of a certain problem and so is the goal for federated learning systems. Therefore algorithms that can optimize the underlying machine learning model or the training process is of interest. There is a trade-off between privacy, especially when utilizing differential privacy, and the model's accuracy [10], thus making algorithms that

can improve the accuracy even more important in a federated learning setting. One suggested approach for the vertical scenario is optimization of the neural networks architecture, by utilizing self-supervised neural architecture search [23].

4.2.4 Training time

In some scenarios the time it takes to train the system can be of importance and thus methods that try to reduce the training time can be interesting. These methods can be connected to improve the computational aspect of the system, by more computationally efficient algorithms or upgrading the computational power at the clients. Another approach that could reduce the training time is to perform each worker update asynchronously, since in a synchronous approach the system will never be faster than the slowest worker [24].

4.2.5 Scalability

It can also be interesting to study how the federated learning system and different methods behave as the number of clients involved in the federation changes [25]. Although this is a more common area in the horizontal case or cross-device setting since here more clients are often available to be added to the system and doing so simply means adding more samples. For the vertical case, which usually is a cross-silo setting, the number of clients is usually pre-determined. Since by adding more clients one has to locate clients with new features of the same samples relevant to the problem. That being said, its still interesting to understand how different vertical federated learning methods behave for different number of clients.

4.3 State of the art vertical federated learning methods

This section summarizes the most interesting state of the art methods that tries to address the problems and areas of improvement presented in the previous section.

4.3.1 FedBCD

Federated Stochastic Block Coordinate Descent (FedBCD) is an algorithm proposed by Liu et al. [21]. The algorithm performs a fixed number of Q local updates between each communication round. The idea is to reduce the number of communication rounds needed to reach convergence, thus addressing the potential bottleneck which the communication can be. To do this, the authors propose an algorithm where each client performs local updates, i.e. repeatedly computes new partial derivatives and updates the parameters, for a fixed number of Q iterations using the last previously known gradient received from the master. After the iterations, the clients can calculate a new intermediate result which it sends to the master who then can calculate new gradients for each respective client and the procedure continues. Since each worker now has to perform more computations, the computational effort in each worker is increased as a trade-off with reducing the number of communication rounds needed for convergence. The authors draws the conclusion from their results that FedBCD significantly reduces the communication overhead as well as the total number of communication rounds needed for convergence, at the cost of computation at the local clients.

4.3.2 Asynchronous training

The standard approach in a vertical federated learning system is a synchronous approach, i.e. the master waits for all workers intermediate result before proceeding with its operations. The synchronous approach can be inefficient due to delays in certain workers, which can be caused by unbalanced computational resources [26] or unreliable connectivity for the communications. This is due to the fact that for a synchronous approach, the federated learning system will never be faster than the slowest worker. In contrast in asynchronous setup, the master never waits for the output from delayed clients, and continues on with the operation as long as it receives input, hence can be considered rather fault-tolerant.

There exists a handful of different asynchronous methods for vertical federated learning, [24], [26], [27], [28]. The main idea across all of them is the same, a worker should be able to query for a new

gradient at any time from the master which then uses the available information it has instead of waiting for the intermediate results from the other workers. This can be done by having the master save previous intermediate results such that when a worker queries for a new gradient the master can utilize the previous intermediate results from the other workers corresponding to the same samples.

4.3.3 Neural architecture search

Liang et al. [23] propose their algorithm SS-VFNAS, self-supervised vertical federated neural architecture search, which incorporates neural architecture search into a vertical federated learning setting with the objective to optimize each clients network structure while maintaining privacy constraints. They incorporate the neural architecture search into a vertical federated learning system such that during training each client also optimizes the structure of their own neural network, self-supervised. From their results they argue that this approach shows superior efficiency and performance at the cost of more computations.

4.3.4 Self-taught federated learning

Chu et al. [29], propose a method where they utilize variational auto encoders (VAE) for unsupervised feature extraction at each client. Their proposed method works by having each client train an encoder as a feature extractor before the training of the entire system starts. The master then only receives the latent variables, which is the output of the encoders, from each client as input to its own machine learning model. The clients encoder is not further trained and thus the clients never receive a gradient from the master. The authors argue that since no gradient is needed to be sent to the clients and that no information can be inferred from the latent variables that no encryption is needed for the messages between the clients and the master. This results in saving a substantial amount of training time and they argue from their results that they achieve this while maintaining a similar accuracy as other methods.

Chapter 5

Problem formulation

The use case at hand is a vertical federated learning problem meaning that the data at the different clients share the same samples but differ in the feature space.

The problem formulation is the following: We assume $\mathcal{K}$ clients and one master that collaboratively learns a model from the data $\{X_i, y_i\}_{i=1}^{N}$ where N is the number of samples. Each client has access to a subset of the features of the data $\{x_i^k \in \mathcal{R}^{d_k}\}_{k=1}^{\mathcal{K}}$ where d_k is the number of features client k has access to. We assume the master has access to labels of all samples, y, but not any input attributes (features). The samples are for the experiments already assumed to be aligned at the clients. The objective of this thesis is to train a collaborative model using SplitNN and applying different state of the art vertical federated learning methods to compare their performance.

5.1 Method

Firstly the current state of the art vertical federated learning methods had to be identified through a literature study. After the current state of the art methods had been outlined, some of them were chosen to be implemented and tested, based on how promising and interesting these methods are. The code is written in Python and is based on a neural network code framework constructed from scratch, which had previously been used in initial trial [30], to give complete freedom and visibility of the neural network to easily be able to implement the changes that the state of the art methods proposes. Each client's code runs in separate pods, each with 5 GB memory and 2 CPU's, at the cluster at Ericsson utilizing Kubernetes. There is open communication links between the clients within the namespace, i.e. a message bus using RabbitMQ. All the clients are connected to RabbitMQ where they listen for packets in this queue. There is a destined source and destination for each package to specify which client should receive each specific message [31].

5.1.1 Data pre-processing

The data used to test the methods, which will be explained in detail in Section 5.2, is divided by a 70/30 split into train and test data. The test set is never exposed to the model during training and is used after training to give an unbiased estimate of the models performance. The data is then put through a Standard Scaler, which removes the mean and scales the data to unit variance. This is done since each feature might be measured in different scales which without scaling the data might result in a bias. Therefore, the Standard Scaler scales each feature individually such that they all have the same mean and variance. No further data pre-processing is performed since the objective of this thesis is not to optimize the results of a specific problem but rather compare different methods applied to the same problem.

5.1.2 Evaluation of results

To evaluate and compare different methods, the metrics f1, precision and recall are used for classification problems. These are based on the confusion matrix, i.e. the number of true positive (TP), true negative

(TN), false positive (FP) and false negative (FN) predictions on a binary classification problem. They are defined as following:

$$precision = \frac{TP}{TP + FP},$$

$$recall = \frac{TP}{TP + FN}$$

and

$$f1 = \frac{2}{1/precision + 1/recall}.$$

For regression problems the R^2 score and mean squared error (MSE) are used to evaluate performance. The R^2 score has a maximum value of 1 for perfectly predicted data and is defined as following:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}},$$

where

$$SS_{res} = \sum (\mathbf{y} - \hat{\mathbf{y}})^2$$

and

$$SS_{tot} = \sum (\mathbf{y} - \bar{\mathbf{y}})^2,$$

where $\bar{\mathbf{y}}$ is the mean of $\mathbf{y}$ and $\hat{\mathbf{y}}$ is the predicted value associated with $\mathbf{y}$, the actual label. And the MSE is defined as following:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y} - \hat{\mathbf{y}})^2$$

These metrics, together with the training time, is logged throughout the training rounds to visualize how the methods behaves during training. After the training is completed, these metrics are also calculated based on the test set, which gives an unbiased estimate of the methods performance.

5.2 Datasets

This section will describe the datasets that were used for experiments.

5.2.1 Video Quality of Experience (QoE)

QoE [32] is a IoT dataset and a binary classification problem with 450 samples and 9 features. The task is to train a model that estimates whether or not a user would experience poor perceived quality. QoE has the following features:

- Temporal complexity index indicating the difference between consecutive displayed video pictures.
- Spatial complexity index indicating the difference between the pixels at the same displayed video picture.
- Displayed video frames per second.
- Number of stalls.
- Initial stalling time before the video start.
- Total stalling time during video stream.
- Average video bitrate.
- The bitrate trend related to bitrate switches indicating whether or not is an increase or decrease.
- Last video playout bitrate.

The representation of QoE is formulated via binarized form collected MOS score in the 1-100 ACR scale with some threshold, where 0 corresponds to poor-or-worse perceived quality and 1 corresponds to good-or-better.

5.2.2 Salesprice

Salesprice is a dataset and a regression problem to try to estimate the salesprice of properties based on features such as the number of fireplaces, number of bedrooms, the square footage etc. There are 9 features and 977 samples. It has the follwing features:

- Number of fireplaces.
- First floor square feet.
- Overall material quality.
- Total rooms above ground.
- Living area above ground.
- Size in square feet.
- Size of garage in square feet.
- Total square feet of basement.
- Bedrooms above basement.

5.3 SplitNN structure

Here the splitNN structure is described for the different scenarios for each dataset. The numbers in Table 5.1 refers to the number of neurons in each layer. Since both datasets have 9 features, in the scenario of two clients they are divided as 4 features in one worker and 5 in the other. For four clients, each client has 2 features except for the final client which has 3. The SplitNN structure is such that the same total number of neural network layers and neurons were used in the experiments for 2 and 4 clients for each respective dataset.

Dataset	Number of clients	workers network	master network
QoE	2	[*input*, 32, 8]	[16, 32, 1]
QoE	4	[*input*, 16, 4]	[16, 32, 1]
salesprice	2	[*input*, 32, 16]	[32, 32, 1]
salesprice	4	[*input*, 16, 8]	[32, 32, 1]

Table 5.1: The SplitNN structure for the different datasets and scenarios.

For the experiments, the activation function applied in every layer is “ReLu”, except for the last layer for classification problems where “Sigmoid” is used to get the output to reflect a probability.

5.4 Tested methods

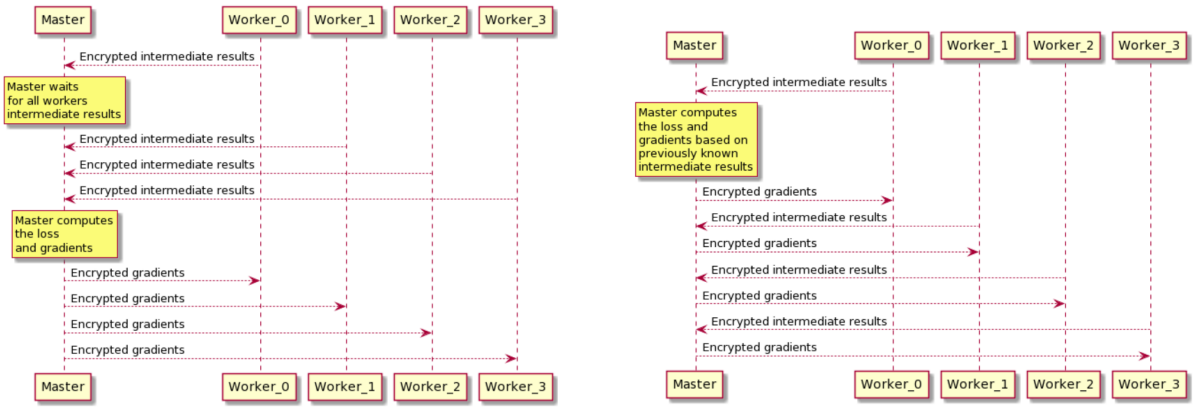
This section describes the different methods that were tested. For each experiment, the result from each method is the average of 10 independent runs to draw confidence. The experiment runs for a predetermined number of rounds, where one round means that one batch of data goes through forward and backpropagation of all the clients in the system and that the parameters are updated.

5.4.1 FedBCD

FedBCD was implemented exactly as described in Section 4.3.1. For the split-NN case this translates to each worker looping over the process of continuing the backpropagation and updating the parameters, for a fixed number of Q iterations. Since the calculation of new partial derivatives during the backpropagation depends on the values of the current parameters, i.e. previously updated model weights, each iteration will calculate new partial derivatives even though they all utilize the last known gradients received from the master. This will hopefully result in fewer communication rounds needed for convergence.

5.4.2 Asynchronous training approach

Inspired by the work of different methods of asynchronous vertical federated learning, an asynchronous approach was developed as a part of the thesis work. The main idea here is that a worker sends its intermediate results and the master uses the last known intermediate result from the other workers corresponding with the correct batch, instead of waiting for the intermediate results from all the other workers. Before training each worker has to send intermediate results of all samples as initialization so that when training starts the master has access to intermediate results corresponding to every worker and sample. Upon receiving new intermediate results, the master overrides the previous intermediate result corresponding to that worker and batch such that the master always has access to the most up to date intermediate result. So basically, for each worker and sample the master saves the intermediate result when received and then utilizes it when another worker sends their intermediate results. The master then combines the newly received intermediate results with the corresponding saved ones from the other workers and then calculates the prediction, loss and starts the backpropagation. The master then sends the gradients only to the worker who sent the intermediate results. An illustration of the differences between a synchronous and asynchronous approach is shown in Figure 5.1.



(a) A synchronous communication scheme.

(b) An asynchronous communication scheme.

Figure 5.1: Illustration of the difference between a synchronous and asynchronous approach.

Due to the nature of the asynchronous approach, the master has to perform more rounds than in the synchronous approach, increased with a factor corresponding to the number of workers in the system. Therefore, the learning rate at the master is divided by the number of workers to make the two methods comparable. This also means that calculated metrics in one round of the asynchronous approach is the average of the metrics from as many rounds as there are workers, to make the two methods results of the training process comparable.

This approach aims to mitigate the effects if one or all of the clients have unreliable connection or slow computational power, and to hopefully converge faster in the case of delays at the workers compared to the synchronous case. This is because in the synchronous case if one worker experiences delays then all the other workers have to wait for it. With an asynchronous approach the other workers can, together with the master, continue with their training instead. Therefore, experiments when the workers is exposed to delays were performed. Two different delay scenarios were introduced, one where each round each worker has a 25% chance of sleeping for five seconds. For the other, called “round-robin delay”, only one worker sleeps for 5 seconds at a time, and upon waking up the next worker starts sleeping.

By utilizing old intermediate result, where it is expected to make the system behave worse than waiting for the other workers’ current data but thus enabling an asynchronous communication scheme, the hypothesis is that in the case of delay scenarios, clear benefits of the asynchronous approach should bee seen as compared to the synchronous.

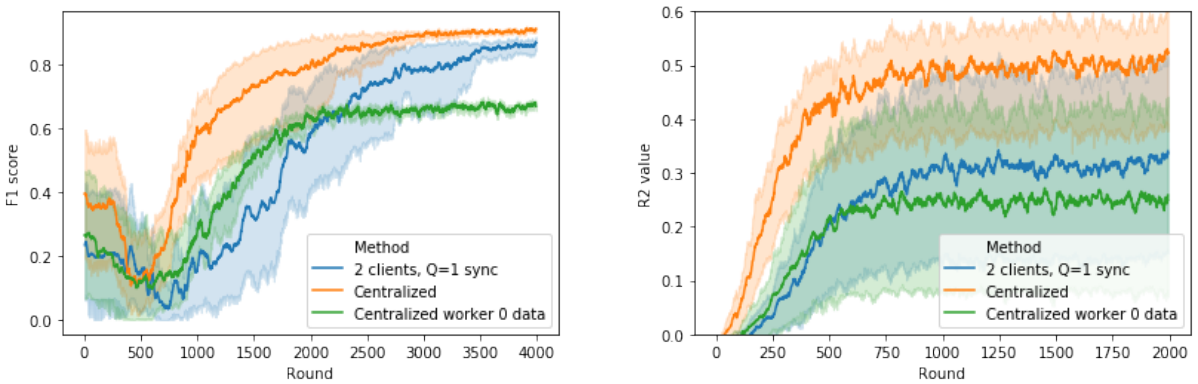
5.4.3 Centralized

To get a performance baseline to compare the different methods with, each dataset is also tested with a centralized method. This is done by having one neural network, with exactly the same number of neurons in each layer as in the respective SplitNN approach. This will then act as the scenario of having complete access to all clients' data, i.e. as if there were no privacy constraints. The difference between the centralized and vertical federated learning approach is that no encryption is needed for the centralized approach. Also, for SplitNN, the neurons across different clients are not connected, due to the privacy constraints of federated learning. This results in the centralized approach having more model parameters than SplitNN, due to the increased amount of links between neurons, if both approaches has the same number of layers and neurons.

Chapter 6

Results

This chapter describes the results of the different experiments performed to evaluate the different methods. For the first experiment, the idea is to illustrate why vertical federated learning is used by comparing three different scenarios. First scenario is an ideal world where you have access to all the data and can train a centralized model. In the two other scenarios we imagine that you only have access to part of the data and you can either train a centralized model on only the data you have access to or you can use vertical federated learning, thus utilizing all the data.



(a) Experiments on the QoE dataset.

(b) Experiments on the Salesprice dataset.

Figure 6.1: F1 score during training for a Centralized approach with access to all the features, a Centralized approach with access only to one clients features and a vFL synchronous approach with 2 clients, on both the QoE and Salesprice datasets.

Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Centralized	0.86 (0.01)	0.83 (0.02)	0.88 (0.02)	65.02 (18.85)
Centralized worker 0 data	0.65 (0.02)	0.62 (0.01)	0.69 (0.04)	68.84 (21.37)
Q=1, 2 clients synchronous	0.83 (0.05)	0.78 (0.05)	0.90 (0.07)	332.51 (58.42)

Table 6.1: Results on the test data after 4000 rounds of training for a Centralized approach with access to all the features, a Centralized approach with access only to one clients features and a vFL approach on the QoE dataset.

Method	R2 (σ)	MSE (σ)	Training time (σ) [s]
Centralized	0.46 (0.26)	0.44 (0.21)	9.60 (0.47)
Centralized worker 0 data	0.22 (0.24)	0.63 (0.20)	9.14 (1.47)
Q=1 2 clients synchronous	0.28 (0.25)	0.58 (0.20)	160.09 (16.96)

Table 6.2: Results on the test data after 2000 rounds of training for a Centralized approach with access to all the features, a Centralized approach with access only to one clients features and a vFL approach on the Salesprice dataset.

The results from Figure 6.1 and Tables 6.1 and 6.2 illustrates the benefits of federated learning. In scenarios where the data is located at different clients and privacy restriction exists, it can often be beneficial to utilize vertical federated learning and thus utilizing all the data as compared to only training a centralized model with the data you have access to. With that said, the vertical federated learning method illustrated in Figure 6.1 is the most "vanilla" case. Although the accuracy of vertical federated learning approaches to the centralized scenario, and is better than the isolated case, the training time (and the number of required epochs) until model convergence is high. Hence, the objective from this point is to improve its results and behaviour by implementing different SOTA vertical federated learning methods.

For the next experiment the FedBCD method was implemented and tested. Figure 6.2 shows the results of the FedBCD method with varying Q values on the QoE dataset using 2 clients and a synchronous approach.

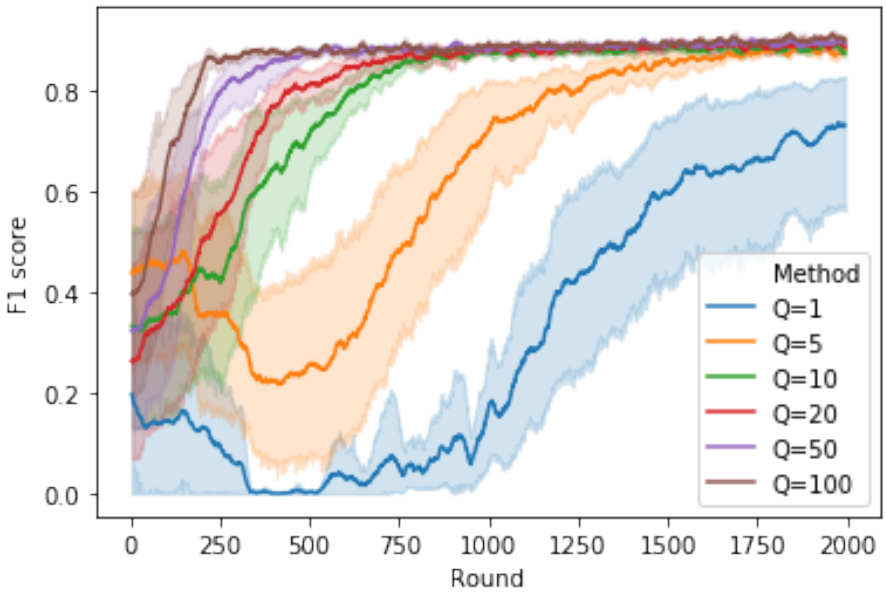


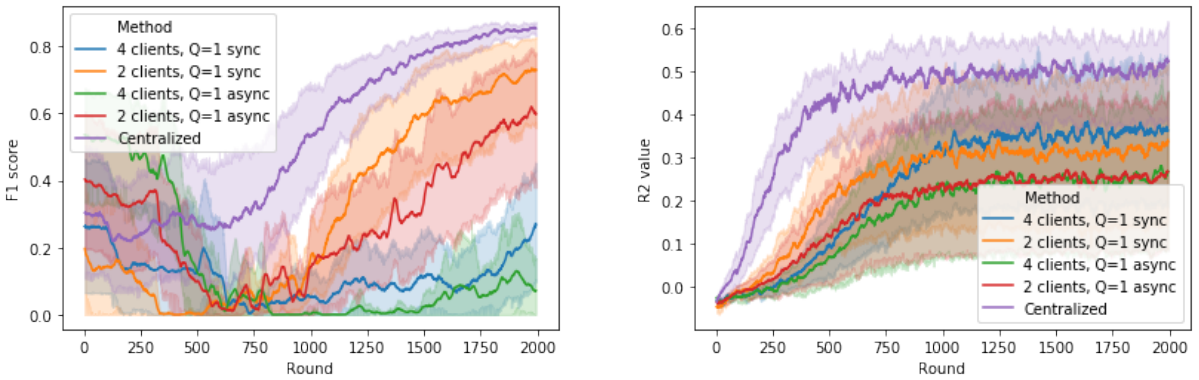
Figure 6.2: F1 score during training for different Q values using the FedBCD method on the QoE dataset.

Q value	rounds	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=1	500	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	61.24 (14.38)
	2000	0.75 (0.22)	0.83 (0.09)	0.75 (0.25)	190.72 (46.02)
Q=5	500	0.17 (0.28)	0.26 (0.37)	0.22 (0.40)	57.08 (10.80)
	2000	0.84 (0.02)	0.80 (0.05)	0.89 (0.03)	181.62 (36.65)
Q=10	500	0.53 (0.29)	0.49 (0.32)	0.69 (0.39)	47.23 (8.02)
	2000	0.86 (0.02)	0.83 (0.03)	0.89 (0.03)	158.11 (37.60)
Q=20	500	0.58 (0.20)	0.70 (0.26)	0.67 (0.31)	60.09 (11.39)
	2000	0.86 (0.02)	0.85 (0.02)	0.88 (0.03)	203.00 (37.55)
Q=50	500	0.52 (0.30)	0.55 (0.37)	0.63 (0.41)	52.55 (8.41)
	2000	0.86 (0.02)	0.83 (0.03)	0.89 (0.03)	183.29 (39.83)
Q=100	500	0.75 (0.09)	0.74 (0.17)	0.82 (0.15)	61.77 (7.87)
	2000	0.86 (0.02)	0.82 (0.04)	0.90 (0.03)	214.12 (33.65)

Table 6.3: Results on the test data of the FedBCD method for different values of Q after 500 and 2000 rounds of training on the QoE dataset.

As seen in Figure 6.2 and Table 6.3, for larger Q values fewer rounds (i.e. communication rounds) is needed for the system to converge. Since each worker performs more computations as Q increases, the expectation is that the training time to increases as well. This does not seem to completely hold and there are apparently other factors that have larger effects on the training time. But by comparing the times at convergence, for example $Q = 100$ at round 500 with $Q = 1$ at round 2000 we can see that the time needed to converge is massively improved, while at the same time maintaining similar testset accuracy scores. This result clearly shows that FedBCD works as intended and achieves its goals of reducing the number of communication rounds needed for convergence. That being said, from the experiments it was found that a upper limit of Q exists and exceeding that limit causes model weight divergence. How large this upper limit is depends on the problem, i.e. the dataset that FedBCD is applied to.

Next, experiments of both the synchronous and asynchronous approaches was performed to study their differences, for both the scenarios of when the data is divided by 2 and 4 clients. Figure 6.3 illustrates the results on both the QoE and Salesprice datasets for the centralized approach and a comparison of the synchronous and asynchronous approach.



(a) Experiments on the QoE dataset.

(b) Experiments on the Salesprice dataset.

Figure 6.3: Training results for the asynchronous and synchronous approaches, with both 2 and 4 clients compared to the centralized approach on two different datasets.

Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Centralized	0.83 (0.02)	0.82 (0.04)	0.85 (0.02)	42.26 (24.90)
Q=1 2 clients asynchronous	0.69 (0.25)	0.69 (0.27)	0.74 (0.30)	186.96 (46.75)
Q=1 2 clients synchronous	0.75 (0.22)	0.83 (0.09)	0.75 (0.25)	190.72 (46.02)
Q=1 4 clients asynchronous	0.09 (0.18)	0.29 (0.47)	0.06 (0.12)	386.82 (81.10)
Q=1 4 clients synchronous	0.28 (0.30)	0.40 (0.43)	0.23 (0.26)	235.55 (30.35)

Table 6.4: Results on the test data after 2000 rounds of training on the QoE dataset.

Method	R2 (σ)	MSE (σ)	Training time (σ) [s]
Centralized	0.46 (0.26)	0.44 (0.21)	9.60 (0.47)
Q=1 2 clients asynchronous	0.23 (0.25)	0.63 (0.21)	131.47 (22.35)
Q=1 2 clients synchronous	0.28 (0.25)	0.58 (0.20)	160.09 (16.96)
Q=1 4 clients asynchronous	0.22 (0.24)	0.63 (0.20)	262.46 (61.70)
Q=1 4 clients synchronous	0.31 (0.22)	0.56 (0.18)	235.31 (30.17)

Table 6.5: Results on the test data after 2000 rounds of training on the salesprice dataset.

As seen in Figure 6.3, we see that 4 clients behave worse than 2 clients which in turn behaves worse than the centralized approach. This can be explained by the fact that since all experiments have exactly the same number of neurons in each layer, the centralized approach has the most parameters, followed by the 2 clients approach and finally the 4 clients approach. This is because for the 2 and 4 clients approaches the neurons in the different clients are not connected, due to the privacy constraints that come with federated learning, thus resulting in fewer parameters for the model to optimize. Figure 6.3 also suggests that the synchronous approach outperforms the asynchronous approach, which is logical since the synchronous approach always uses the most updated intermediate results while the asynchronous approach may use stale information. These tendencies are also seen in the results of the testset data in Tables 6.4 and 6.5.

We can also now apply the proven FedBCD method to both the synchronous and asynchronous approaches for both datasets with either 2 or 4 clients.

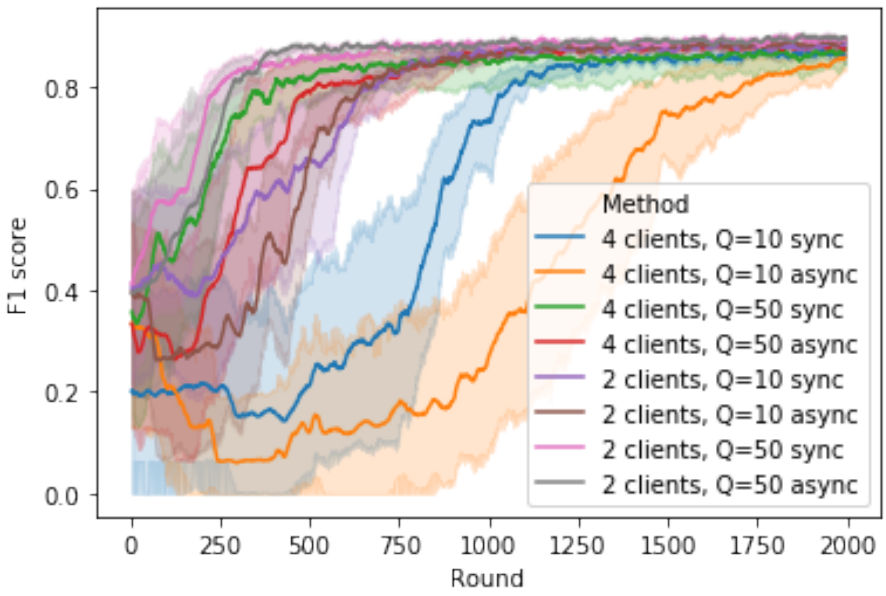


Figure 6.4: F1 score during training for different Q values for both 2 and 4 clients and synchronous and asynchronous approaches on the QoE dataset.

Method	rounds	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=10 2 clients async	500	0.56 (0.31)	0.62 (0.35)	0.55 (0.34)	59.97 (10.69)
	2000	0.86 (0.02)	0.84 (0.01)	0.89 (0.04)	201.91 (43.53)
Q=10 2 clients sync	500	0.68 (0.26)	0.78 (0.29)	0.64 (0.28)	60.42 (8.63)
	2000	0.87 (0.01)	0.84 (0.02)	0.89 (0.02)	198.31 (32.11)
Q=10 4 clients async	500	0.14 (0.26)	0.26 (0.44)	0.14 (0.30)	94.07 (15.57)
	2000	0.86 (0.05)	0.82 (0.07)	0.90 (0.04)	331.30 (58.24)
Q=10 4 clients sync	500	0.24 (0.38)	0.36 (0.47)	0.23 (0.39)	75.09 (12.69)
	2000	0.87 (0.01)	0.84 (0.02)	0.90 (0.01)	231.06 (35.29)
Q=50 2 clients async	500	0.85 (0.02)	0.82 (0.02)	0.89 (0.02)	63.81 (15.58)
	2000	0.86 (0.01)	0.83 (0.03)	0.90 (0.01)	204.32 (42.48)
Q=50 2 clients sync	500	0.83 (0.06)	0.82 (0.07)	0.83 (0.08)	73.73 (14.45)
	2000	0.86 (0.01)	0.84 (0.04)	0.89 (0.02)	248.19 (58.51)
Q=50 4 clients async	500	0.80 (0.14)	0.85 (0.05)	0.80 (0.22)	90.31 (16.70)
	2000	0.87 (0.02)	0.84 (0.04)	0.90 (0.03)	318.77 (58.84)
Q=50 4 clients sync	500	0.84 (0.07)	0.82 (0.10)	0.88 (0.09)	88.13 (11.25)
	2000	0.86 (0.05)	0.84 (0.06)	0.88 (0.05)	289.09 (47.86)

Table 6.6: Results on the test data after 500 and 2000 rounds of training on the QoE dataset.

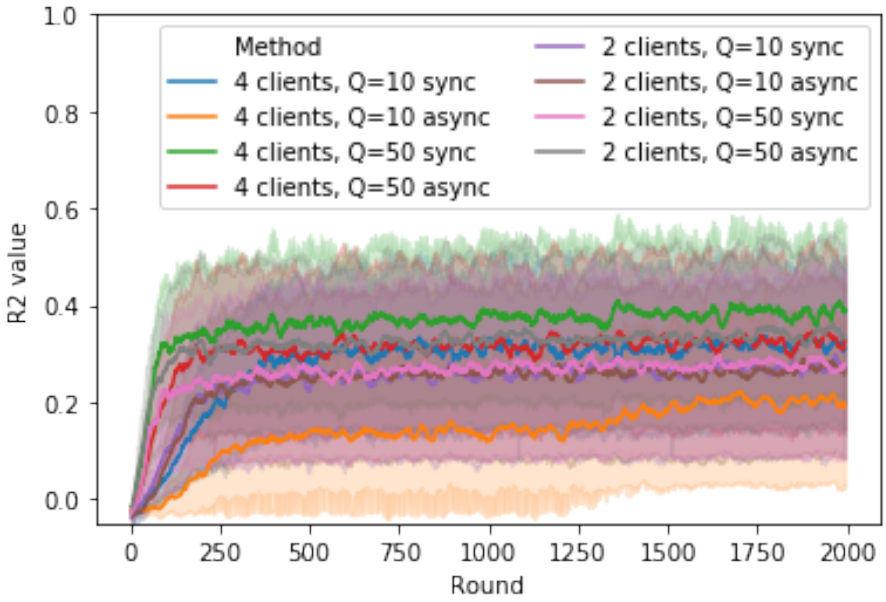


Figure 6.5: R2 values during training for different Q values for both 2 and 4 clients and synchronous and asynchronous approaches on the salesprice dataset.

Method	rounds	R2 (σ)	MSE (σ)	Training time (σ) [s]
Q=10 2 clients async	500	0.22 (0.24)	0.63 (0.20)	46.58 (6.23)
	2000	0.26 (0.28)	0.60 (0.23)	141.70 (18.97)
Q=10 2 clients sync	500	0.21 (0.23)	0.64 (0.19)	54.48 (7.21)
	2000	0.26 (0.28)	0.60 (0.23)	172.81 (12.52)
Q=10 4 clients async	500	0.12 (0.20)	0.72 (0.16)	64.23 (8.93)
	2000	0.19 (0.25)	0.66 (0.20)	208.91 (30.27)
Q=10 4 clients sync	500	0.25 (0.22)	0.61 (0.18)	66.74 (4.74)
	2000	0.29 (0.25)	0.58 (0.21)	222.07 (19.01)
Q=50 2 clients async	500	0.28 (0.25)	0.59 (0.20)	53.12 (3.16)
	2000	0.33 (0.29)	0.54 (0.24)	167.22 (10.07)
Q=50 2 clients sync	500	0.24 (0.26)	0.62 (0.21)	58.51 (2.67)
	2000	0.28 (0.30)	0.59 (0.24)	191.31 (9.02)
Q=50 4 clients async	500	0.28 (0.25)	0.59 (0.20)	72.63 (7.84)
	2000	0.32 (0.28)	0.56 (0.23)	236.47 (27.27)
Q=50 4 clients sync	500	0.30 (0.22)	0.57 (0.18)	67.70 (6.94)
	2000	0.36 (0.26)	0.52 (0.21)	224.82 (29.42)

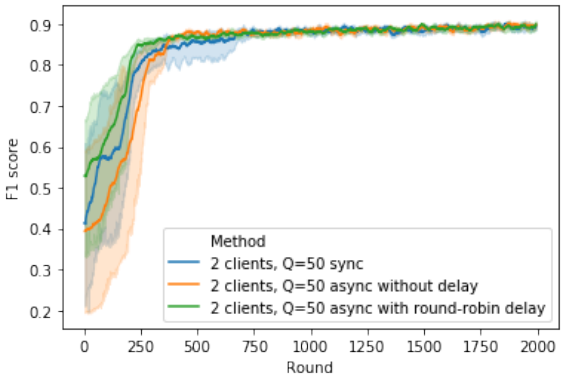
Table 6.7: Results on the test data after 500 and 2000 rounds of training on the salesprice dataset.

From the results in Figures 6.4 and 6.5 we again can see the effects of FedBCD, i.e. convergence on fewer communication rounds. We can also see from Tables 6.6 and 6.7 that the asynchronous approach performs slightly worse compared to the synchronous approach in terms of both test accuracy and training time, but often has very comparable results to the synchronous approach.

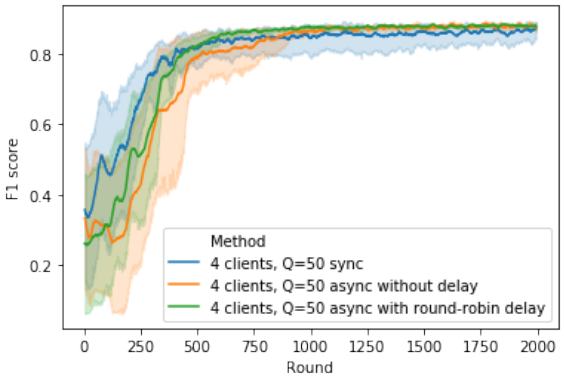
Here we can also see interesting results on the training time. With the asynchronous approach, the master has to perform one round for each of the workers' rounds, so in the 2 clients asynchronous approach the master performs 4000 rounds (2000 rounds for each worker) and in the 4 clients asynchronous approach the master performs 8000 rounds. This results in that the master has to perform more com-

munication and computations which subsequently results in a longer execution time. On the other hand, the benefit of an asynchronous approach is that the master does not have to wait for the slowest worker. Here one can see that for two clients these effects almost cancel each other out with similar training times for the synchronous and asynchronous approaches. But for four clients, the extra communication and computations the master needs to do takes over.

Based on the previous results and the fact that an asynchronous approach main advantage is its ability to continue the training in case of failure or delay at one worker, we now want to test these methods with an artificial delay implemented compared to the synchronous and asynchronous approaches without any delay. The first delay scenario called "round-robin delay", where one worker sleeps for 5 seconds and upon waking up the next worker starts to sleep and so on. The results from this delay scenario is shown in Figure 6.6 and Tables 6.8 and 6.9.



(a) Asynchronous approach exposed to round-robin delay with 2 clients.



(b) Asynchronous approach exposed to round-robin delay with 4 clients.

Figure 6.6: Training results for the asynchronous and synchronous approaches without delays and the asynchronous approach exposed to round-robin delay with both 2 and 4 clients on the QoE dataset.

Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=50 2 clients sync	0.83 (0.06)	0.82 (0.07)	0.83 (0.08)	73.73 (14.45)
Q=50 2 clients async	0.85 (0.02)	0.82 (0.02)	0.89 (0.02)	63.81 (15.58)
Q=50 2 clients async delay	0.85 (0.03)	0.83 (0.07)	0.88 (0.05)	71.90 (6.90)

Table 6.8: Results on the test data with 2 clients after 500 rounds of training on the QoE dataset of the asynchronous approach with round-robin delay compared to both the synchronous and asynchronous approach without delays.

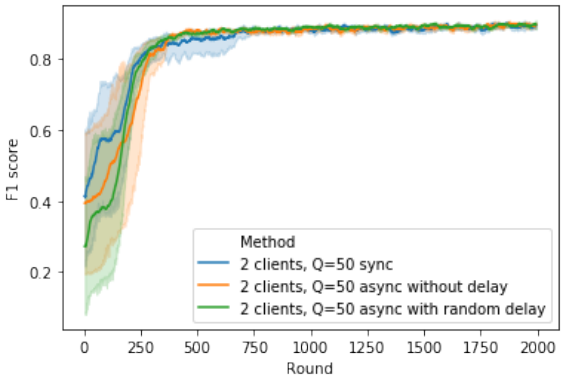
Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=50 4 clients sync	0.84 (0.07)	0.82 (0.10)	0.88 (0.09)	88.13 (11.25)
Q=50 4 clients async	0.80 (0.14)	0.85 (0.05)	0.80 (0.22)	90.31 (16.70)
Q=50 4 clients async delay	0.84 (0.04)	0.81 (0.05)	0.88 (0.06)	68.74 (10.69)

Table 6.9: Results on the test data with 4 clients after 500 rounds of training on the QoE dataset of the asynchronous approach with round-robin delay compared to both the synchronous and asynchronous approach without delays.

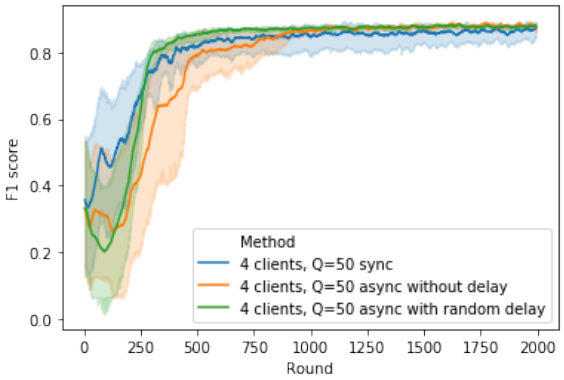
From the results in Figure 6.6 and Table 6.8 and 6.9 we see that the asynchronous approach when exposed to the round-robin delay scenario behaves very comparable to both the asynchronous and synchronous approaches without any delay. The main takeaway from here is the training time. For com-

parison, if the synchronous approach were to be exposed to the same round-robin delay scenario, then one worker would sleep for 5 seconds each round while the other workers would have to wait for it. This would result in a training time of at least $5 \cdot 500 = 2500$ seconds after 500 rounds. Instead for the asynchronous approach the time after 500 rounds were 71.90 and 68.74 seconds for 2 and 4 clients respectively. Therefore these results clearly illustrate the benefits of using an asynchronous approach in a scenario where delays could possibly occur.

For the second delay scenario, each worker now instead has a 25% chance of sleeping each round regardless of the other workers status.



(a) Asynchronous approach exposed to random delay with 2 clients.



(b) Asynchronous approach exposed to random delay with 4 clients.

Figure 6.7: Training results for the asynchronous and synchronous approaches without delays and the asynchronous approach exposed to random delays with both 2 and 4 clients on the QoE dataset.

Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=50 2 clients sync	0.83 (0.06)	0.82 (0.07)	0.83 (0.08)	73.73 (14.45)
Q=50 2 clients async	0.85 (0.02)	0.82 (0.02)	0.89 (0.02)	63.81 (15.58)
Q=50 2 clients async delay	0.85 (0.01)	0.81 (0.04)	0.88 (0.05)	733.92 (209.66)

Table 6.10: Results on the test data with 2 clients after 500 rounds of training on the QoE dataset of the asynchronous approach with random delay compared to both the synchronous and asynchronous approach without delays.

Method	f1 (σ)	precision (σ)	recall (σ)	Training time (σ) [s]
Q=50 4 clients sync	0.84 (0.07)	0.82 (0.10)	0.88 (0.09)	88.13 (11.25)
Q=50 4 clients async	0.80 (0.14)	0.85 (0.05)	0.80 (0.22)	90.31 (16.70)
Q=50 4 clients async delay	0.86 (0.02)	0.83 (0.05)	0.90 (0.02)	678.74 (21.70)

Table 6.11: Results on the test data with 4 clients after 500 rounds of training on the QoE dataset of the asynchronous approach with random delay compared to both the synchronous and asynchronous approach without delays.

From Figure 6.7 and Table 6.10 and 6.11 we can see that for this delay experiment we still see a massive increase in the training time needed for convergence. This is explained by the relative large chance for a worker to sleep, resulting in it being common for every worker to sleep at the same time. On the other hand, both in test score and with the training curve, the asynchronous approach with delay behaves comparably or even slightly outperforms the other approaches.

Chapter 7

Discussion

From the results, we can immediately conclude that FedBCD is an easily implemented but still powerful method to reduce the number of rounds needed for convergence. The experiments clearly show a reduction in the number of training rounds needed for convergence, at the cost of more computations at each of the workers. From the experiments it can also be concluded that a too large Q may cause weight divergence, as Liu et al. [21] also mentions. In other words, the optimal value of Q is problem dependent.

Another conclusion is that the performance worsen when going from centralized approach to vertical federated learning with 2 clients and finally 4 clients. This is expected since in each step the number of parameters decreases. But the experiments with 2 and 4 clients should not be compared to each other. Instead, their purpose here is to give another dimension to investigate and compare the different implemented methods. This is because in a real application scenario, the data is located wherever it is under privacy constraints and there would never be a further split of the available data into more clients as done in these experiments. One can also note that the experiments on the salesprice dataset have much higher variance as compared to the QoE dataset. Therefore one has to be cautious to draw conclusions from the experiments on the salesprice dataset alone, although from all the experiments the results are consistent for both datasets, even with the high variance of the salesprice dataset.

In general a centralized approach is the best approach if all the data is accessible, which is not the case in vertical federated learning, where data is under privacy constraints. But vertical federated learning is still better than only using part of the data. This is shown in Figure 6.1, where if only training a centralized model with the data one worker has access to, it is possible to improve the result by a vertical federated learning approach which utilizes all the features.

Also depending on the problem setting, an asynchronous approach seems preferable due to its similar results to the synchronous approach in the experiments without delay together with its ability to continue the training in case of delays or communication problems, as shown with the round-robin delay scenario. From this experiment, even though there is always one worker sleeping, it does not negatively affect neither the training time nor the testset accuracy scores. If anything, it seems to slightly improve those areas. As seen in Table 7.1 the asynchronous and synchronous approaches behaves comparably in a scenario without added delays, where for two clients the asynchronous approach behaves slightly better in terms of testset accuracy scores and training time. For four clients the synchronous approach behaves slightly better. The main takeaway is the fault tolerance of the asynchronous approach as shown in the delay experiments. The fault tolerance can be an important ability to have in a method in a real application scenario and since the synchronous and asynchronous approaches displayed comparable results in experiments without added delays, it makes the asynchronous approach the preferable choice in a real application scenario.

Method	Clients	Fault tolerance	f1 (σ)	Training time (σ) [s]
Q=50 Synchronous	2	*No*	0.83 (0.06)	73.73 (14.45)
	4	*No*	0.84 (0.07)	88.13 (11.25)
Q=50 Asynchronous	2	*Yes*	0.85 (0.02)	63.81 (15.58)
	4	*Yes*	0.80 (0.14)	90.31 (16.70)

Table 7.1: Comparison of the synchronous and asynchronous approaches after 500 rounds of training without added delays on the QoE dataset.

It is also interesting to notice that the experiments with delays from Figure 6.7 seems to converge faster than those without in regards to the number of training rounds. This is probably due to the relative high chance for each worker to sleep for 5 seconds, resulting in a high probability of several or all worker sleeping at the same time. In the case of all workers sleeping at the same time, then the training pauses, which we can tell from the longer training times in Tables 6.10 and 6.11. If only several worker sleeps at the same time, thus leaving at least one worker available to continue the training, the federated learning system seems to be able to improve its performance when only focusing on a single or couple of workers at a time, which can be seen in Figure 6.7b where the asynchronous approach with random delay converges faster, in terms of rounds, than the methods without delay.

Chapter 8

Concluding remarks

8.1 Conclusions

Since data privacy is becoming a more and more relevant topic and the continuous development of machine learning and artificial intelligence, vertical federated learning will probably be implemented in increasingly more different areas and more advanced state of the art method will be developed. This thesis has illustrated the need as well as the promising results of vertical federated learning along with its current state of the art methods. From the experiments conducted, FedBCD and an asynchronous approach displayed the best performance. For FedBCD it was in terms of reducing the number of communication rounds needed for convergence and thus reducing the communication overhead. For the asynchronous approach it was in terms of its ability to continue training in case of delays, displaying a similar performance when exposed to delays as compared to scenarios without delays. That being said, in a perfect world where delays never would occur, the synchronous approach slightly outperforms the asynchronous. The real world is seldom perfect though and thus, an asynchronous approach would probably behave better in a real application scenario due to its ability to continue training in case of delays.

8.2 Future work

Since this study is a comparison of the available state of the art methods in vertical federated learning, a more thorough comparison study could be performed if more methods were applied. This opens the possibility for further work, where for example the methods "Neural architecture search" and "Self-taught federated learning" mentioned in Section 4.3 also could be implemented and compared to the methods that were implemented in this thesis. Since vertical federated learning is an active research area, new state of the art methods are frequently introduced. This thesis can then potentially act as the groundwork for further comparisons of both existing and potential future state of the art methods in vertical federated learning.

Bibliography

[1] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[2] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[3] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[5] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[6] Apple WWDC. Apple. designing for privacy (video and slide deck), 2019. https://developer.apple.com/ videos/play/wwdc2019/708.

[7] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection, 2021.

[8] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.

[9] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[10] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[11] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[12] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

[13] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[14] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[15] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

[16] Iker Ceballos, Vivek Sharma, Eduardo Mugica, Abhishek Singh, Alberto Roman, Praneeth Vepakomma, and Ramesh Raskar. Splitnn-driven vertical partitioning. *arXiv preprint arXiv:2008.04137*, 2020.

[17] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[18] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

[19] Oscar Li, Jiankai Sun, Xin Yang, Weihao Gao, Hongyi Zhang, Junyuan Xie, Virginia Smith, and Chong Wang. Label leakage and protection in two-party split learning. *arXiv preprint arXiv:2102.08504*, 2021.

[20] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[21] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. A communication efficient collaborative learning framework for distributed features, 2020.

[22] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[23] Xinle Liang, Yang Liu, Jiahuan Luo, Yuanqin He, Tianjian Chen, and Qiang Yang. Self-supervised cross-silo federated neural architecture search, 2021.

[24] Ming Li, Yiwei Chen, Yiqin Wang, and Y Pan. Efficient asynchronous vertical federated learning via gradient prediction and double-end sparse compression. In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 291–296. IEEE, 2020.

[25] Latif U Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *arXiv preprint arXiv:2009.13012*, 2020.

[26] Qingsong Zhang, Bin Gu, Cheng Deng, and Heng Huang. Secure bilevel asynchronous vertical federated learning with backward updating. *arXiv preprint arXiv:2103.00958*, 2021.

[27] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. Privacy-preserving asynchronous federated learning algorithms for multi-party vertically collaborative learning. *arXiv preprint arXiv:2008.06233*, 2020.

[28] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*, 2020.

[29] Kai-Fung Chu and Lintao Zhang. Privacy-preserving self-taught federated learning for heterogeneous data. *arXiv preprint arXiv:2102.05883*, 2021.

[30] Selim Ickin. Ericsson, Personal Communication.

[31] Selim Ickin, Konstantinos Vandikas, and Markus Fiedler. Privacy preserving qoe modeling using collaborative learning. In *Proceedings of the 4th Internet-QoE Workshop on QoE-Based Analysis and Management of Data Communication Networks*, Internet-QoE'19, page 13–18, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Zhengfang Duanmu, Abdul Rehman, and Zhou Wang. A quality-of-experience database for adaptive video streaming. *IEEE Transactions on Broadcasting*, 64(2):474–487, June 2018.