Kingdom of Saudi Arabia
Ministry of Education
King Abdulaziz University
Faculty of Computing and Information Technology
Department of Information Systems

المملكة العربية السعودية
وزارة التعليم
جامعة الملك عبد العزيز
كلية الحاسبات وتقنية المعلومات
قسم نظم المعلومات

Beyond Conventional Aggregation: Exploring Asynchronous and Compressed Methods in Federated Learning

Assim Marwan Alharbi ID :1935407
Mohammed Osama Katib ID :1846180
Omar Abdulwahab Khogir ID :1408368

Supervisor

Dr. Yousef Saeed Alsenani

Abstract

The learning technique called federated learning was first proposed by Google in 2017, the goal was to solve a few problems facing the traditional machine learning techniques, Since the day federated learning was published; a large number of aggregation methods have been proposed from different researches, Each of these methods achieved an accuracy result under specific conditions and with different type of data. We review and examine state of art and recent research papers in the federated learning field, whether it's papers proposing a new aggregation method, or papers conducting a survey or performance evaluation of number of these methods. Federated learning brought several number of unique challenges such as data heterogeneity, communication between clients and sever, and security and privacy adaption. There are well-researched aggregation algorithm in literature to solve each of these challenges independently. Therefore, in this project we first study the impact of each challenges, second, we present and analyze the well-known methods, then we propose asynchronous and compressed model that address number of challenges . Finally, experimental is conducted to benchmark the trade-off between the accuracy and communication, and between the accuracy and privacy.

Contents

List of Figures

List of Tables

Chapter 1

Project Outlines

1.1 Introduction

For the last two decades computer science has been in a revolution. Nowadays we see more computer programs learn, like software that adapt its behavior to match the requirements of its tasks. We also have programs that learn to recognize human faces, understand speech, drive cars, and recommend what movie to watch. All this and more to come in the future.

Since machine learning became a big part of the world that we live in, a lot of measures needed to be taken towards it, it relays on data that are provided by devices - whether it's mobile phones, IoT devices or others -, that caused a lot of concern about the privacy of the users that are using these devices. So restricted rules were issued towards the use and sharing of the private data that are collected from these devices.

In 2017 Google came up with a good solution to this problem, they called it Federated Learning[1], rather than storing the data that the users sent to the server and perform the machine learning algorithms on, the model is sent to the users' devices and the model gets trained locally on their devices without any data being shared.

A lot of aggregation methods have been proposed since then, some of it enhanced on an already existing method, while others developed a new aggregation method. The goal of this research is to first introduce these methods in detail to understand the logic behind them, then putting them into testing using a federated learning framework, to see which method produce the best results under certain conditions.

1.2 Problem Statement

Since the day federated learning was published; a large number of aggregation methods have been proposed from different researches, As a new field, it has many challenges like heterogeneous. Each of these methods designed to tackle individual challenge and achieve an accuracy result under specific conditions with different type of data and data distribution. To know which method is optimal under which circumstances; we have to have a full examination on these method and the different scenarios.

1.3 Clear Statement of Objectives

1. Reviewing the state-of-art methods in federated learning and explain the logic behind each of the algorithms without going in much detail about the equations.

2. Comparing The Algorithms: We will compare all the algorithms of federated learning and study their work and the accuracy that they achieved by using Python framework called PyTorch.

3. Propose a model that address communication overhead, asynchronous training, and compression of data.

1.4 Methodology

We review state of art research and recent research in federated learning.

1.5 Project Plan

1.5.1 Task specification

Seeing as how we picked a research topic, we divided task specification into four components: design, research, implementation, and testing.

Design:

- Choice of topic
- Division of labor

Research:

- Collecting data
- Understanding data
- Gathering documentation
- Data analysis

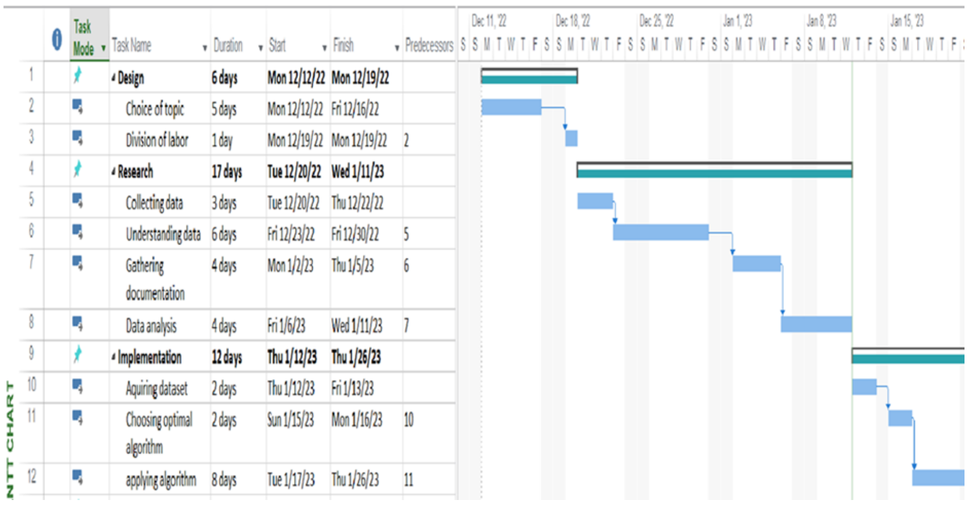
Implementation:

- Acquiring data set
- Choosing optimal algorithm
- Applying algorithm

Testing:

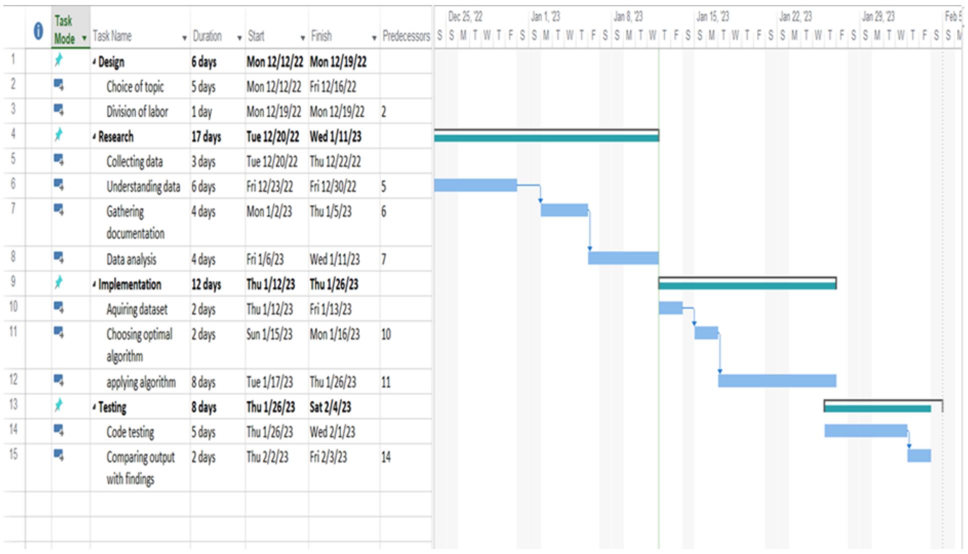
- Code testing
- Ensuring output matches our findings

1.5.2 Task duration (Gantt Chart)



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Design**	**6 days**	**Mon 12/12/22**	**Mon 12/19/22**	
2		Choice of topic	5 days	Mon 12/12/22	Fri 12/16/22	
3		Division of labor	1 day	Mon 12/19/22	Mon 12/19/22	2
4		**Research**	**17 days**	**Tue 12/20/22**	**Wed 1/11/23**	
5		Collecting data	3 days	Tue 12/20/22	Thu 12/22/22	
6		Understanding data	6 days	Fri 12/23/22	Fri 12/30/22	5
7		Gathering documentation	4 days	Mon 1/2/23	Thu 1/5/23	6
8		Data analysis	4 days	Fri 1/6/23	Wed 1/11/23	7
9		**Implementation**	**12 days**	**Thu 1/12/23**	**Thu 1/26/23**	
10		Aquiring dataset	2 days	Thu 1/12/23	Fri 1/13/23	
11		Choosing optimal algorithm	2 days	Sun 1/15/23	Mon 1/16/23	10
12		applying algorithm	8 days	Tue 1/17/23	Thu 1/26/23	11

Figure 1.1: Project Plan 1



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Design**	**6 days**	**Mon 12/12/22**	**Mon 12/19/22**	
2		Choice of topic	5 days	Mon 12/12/22	Fri 12/16/22	
3		Division of labor	1 day	Mon 12/19/22	Mon 12/19/22	2
4		**Research**	**17 days**	**Tue 12/20/22**	**Wed 1/11/23**	
5		Collecting data	3 days	Tue 12/20/22	Thu 12/22/22	
6		Understanding data	6 days	Fri 12/23/22	Fri 12/30/22	5
7		Gathering documentation	4 days	Mon 1/2/23	Thu 1/5/23	6
8		Data analysis	4 days	Fri 1/6/23	Wed 1/11/23	7
9		**Implementation**	**12 days**	**Thu 1/12/23**	**Thu 1/26/23**	
10		Aquiring dataset	2 days	Thu 1/12/23	Fri 1/13/23	
11		Choosing optimal algorithm	2 days	Sun 1/15/23	Mon 1/16/23	10
12		applying algorithm	8 days	Tue 1/17/23	Thu 1/26/23	11
13		**Testing**	**8 days**	**Thu 1/26/23**	**Sat 2/4/23**	
14		Code testing	5 days	Thu 1/26/23	Wed 2/1/23	
15		Comparing output with findings	2 days	Thu 2/2/23	Fri 2/3/23	14

Figure 1.2: Project Plan 2

Chapter 2

Literature Review

2.1 Background

2.1.1 AI and Machine Learning

What is Artificial Intelligence? The Oxford English Dictionary defines the artificial intelligence as: “The theory and development of computer systems able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.” [2]

To develop an AI in a computer system we need to know more about its subsets, we need to understand how it became intelligence.

One of those subsets is known as Machine Learning, it’s the process of using mathematical models of data to help the computer learn without direct instruction. This enables a computer system to continue learning and improving on its own, based on experience.

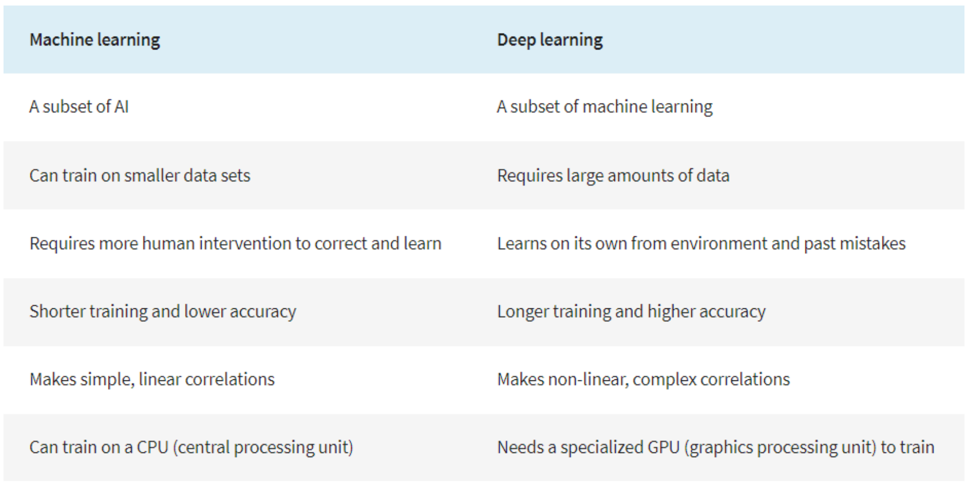
The AI and the machine learning work together to make the computer intelligence using these steps:

- An AI system is built using machine learning and other techniques.
- Then machine learning models are created by studying patterns in the data.
- The data scientists optimize the machine learning models based on patterns in the

data.

- Finally, the process repeats and is refined until the models' accuracy is high enough for the tasks that need to be done.

Another part of AI is called Deep Learning, to know more about the difference between machine Learning and deep learning take a look at these key differences:



Machine learning	Deep learning
A subset of AI	A subset of machine learning
Can train on smaller data sets	Requires large amounts of data
Requires more human intervention to correct and learn	Learns on its own from environment and past mistakes
Shorter training and lower accuracy	Longer training and higher accuracy
Makes simple, linear correlations	Makes non-linear, complex correlations
Can train on a CPU (central processing unit)	Needs a specialized GPU (graphics processing unit) to train

Figure 2.1: Machine Learning vs Deep Learning

2.1.2 Federated Learning

The learning technique called federated learning was first proposed in the article "Communication-Efficient Learning of Deep Networks from Decentralized Data"[1] by Google in 2017, the goal was to solve a few problems facing the traditional machine learning techniques, one of those problems was the amount of data that needs to be centrally stored, another problem was the private data that could not been collected due to certain regulations.

The proposed solution stated that, users (also referred to as clients) which are coordinated by a central server. Each of them has his own training data which is never uploaded to the server. Instead, clients get a randomized global model which is generated at the server side, and then train that model on the local data. after that the local models are sent back to the server, the server aggregates the models into a new global model, then again that model is sent back to the clients, the cycle keeps on repeating until we have a more accurate model.

2.1.3 Characteristics of Federated Learning

Federated learning has many characteristics that makes it differ from traditional learning, these characteristics are: Model is trained on distributed data, The data stays with its owner and not shared with the server, the training happens cross-device with multiple devices participating, it can operate on heterogeneous data, It has high user data privacy.

2.1.4 Federated Learning Architecture

The figure below shows the architecture of the federated learning.

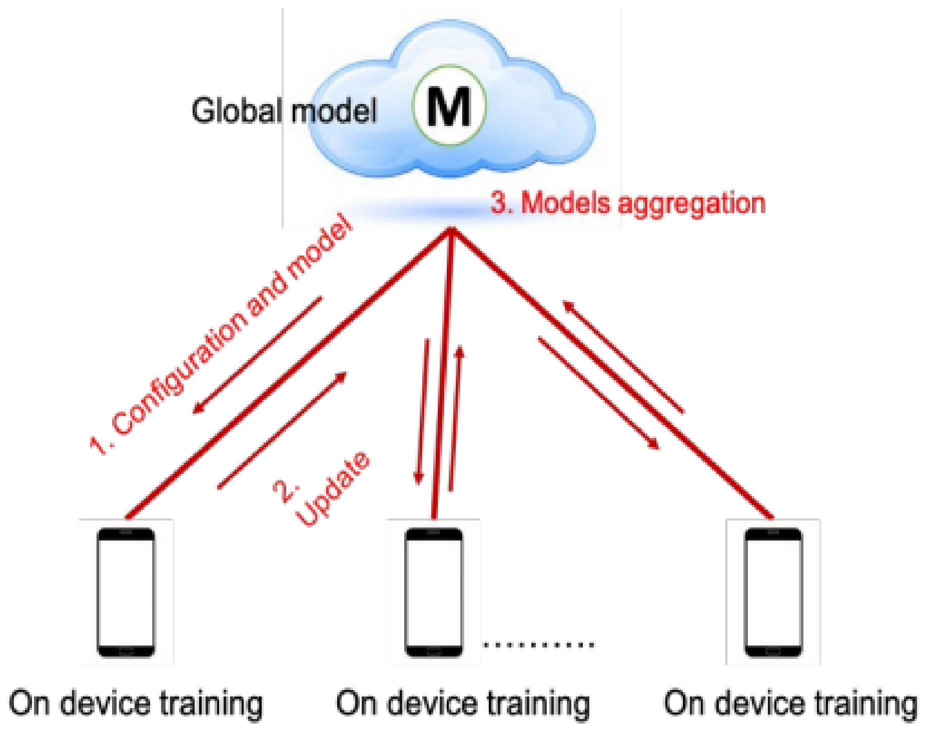


Figure 2.2: Federated Learning Architecture

2.1.5 Privacy

Federated learning has a big advantage on privacy compared to data center training on persisted data. the models updates that are sent by the clients will never contain information more than the raw training data, also the source of that update is not needed by the aggregation algorithm.

2.1.6 Methods

A lot of methods have been published over the past few years; the first method was FedAvg which was introduced in the first publications related to federated learning. Another method that has been proposed is called FedPer, an algorithm incorporating a base and personalized layer and transfer learning methodologies, another method is the FedMA.

In this section we well introduce a number of the federated learning methods with a brief description.

2.1.6.1 FedAvg

Federated Averaging or FedAvg, a global model aggregation algorithm that gets executed by the server once it receives the local updates from clients, it was introduced in the first publications related to federated learning [1].

What is shown below is a simple figure that shows how to get the average of the weight matrix sent by the clients.

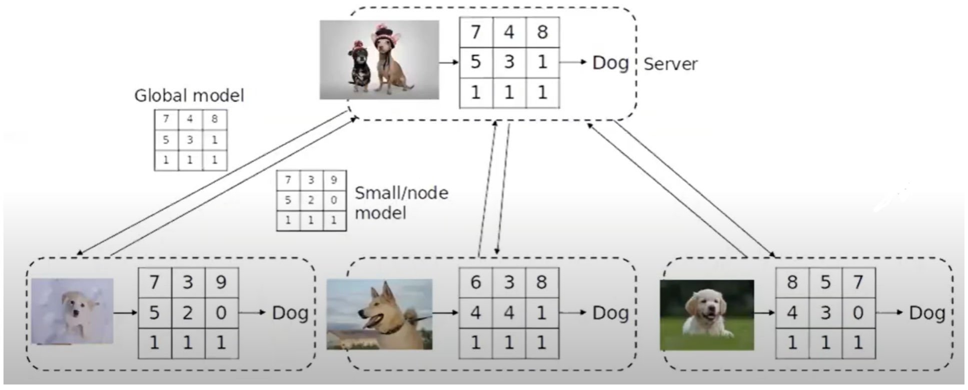


Figure 2.3: FedAvg example

2.1.6.2 FedProx

FedProx was published in the article "Federated Optimization in Heterogeneous Networks"[3], it enhances on FedAvg to make it more suitable to heterogeneous data.

2.1.6.3 FedMA

FedMa (Federated Matched Averaging)[4]: it's an algorithm that designed of convolutional neural networks (CNNs) and, Long Short-Term memory (LSTMs).

2.1.6.4 FedPer

FEDPER is another federated learning algorithm that consists of two later, a base layer and a personalization layer, this method aims to avoid and solve problematic consequences of statistical heterogeneity, in this method and through the study effects of personalization setup on federated learning, it is believed the personalization a key factor in machine learning, but is also achievable due to peoples different personalities and opinions.

2.1.6.5 FedSGD

FedSGD (Federated Stochastic Gradient Descent): is to move this algorithm to the federated setting. The gradients average by the server proportionally to the number of training samples on each node, to make a gradient descent step.

A random client calculates a single batch gradient is done per round of communication but requires very large number of round of training to results good models.

2.1.6.6 FedPAQ

FedPAQ is one of many federated learning algorithms which is communication-oriented based techniques, this novel algorithm uses three techniques and methods and aims to resolve two of the main challenges which federated learning faces using them, these two challenges mentioned are communication bottle-neck and scale.

1. Firstly, communication bottle-neck. Which entails that the flow of data was impaired or stopped entirely, communication bottle-neck can occur due to many factors, some of which are[5]:

• The massive amount of devices that are communicating.

• End-users network connections are slower than that of a data center.

To better enhance communication factors and make it more efficient, Quantization and Sparsification are two model compression techniques that are combined with the averaging algorithms in order to make the communications better.

2. Secondly, Scale. Usually in federated learning the network is comprised of a large number of devices that may range up from thousands all the way to millions of devices, statues of these devices may hinder the federated learning network if they are slow or if they are absolutely inactive.

2.2 Overview of related work

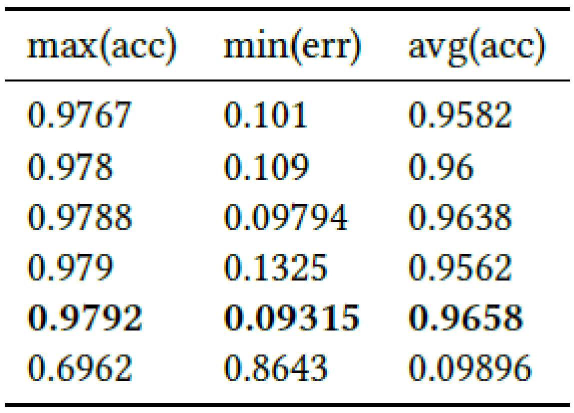
In this section we will introduce the papers or articles that did comparison between multiple aggregation methods, or provided topic or content that is related to this research.

In the paper[6], they benchmarked three federated learning algorithms and compare their performance against a centralized approach where data resides on the server. The three algorithms Federated Averaging (FedAvg), Federated Stochastic Variance Reduced Gradient, and CO-OP are evaluated on the MNIST dataset, using both i.i.d. partitioning and non-i.i.d. partitioning of the data.

The process to create the non-i.i.d. partitioning with folds is as follows: Shuffle the dataset, divide it into five folds, sort each fold, divide each fold into 200 shards of size 70, uniquely assign two random shard positions to each client. Finally, Assign two shards to each client at its shard positions from each fold

the results show that FedAvg achieves the highest accuracy among the federated algorithms, regardless of how data was partitioned. their comparison between FedAvg and centralized learning shows that they are practically equivalent when i.i.d. data is used. However, the centralized approach outperforms FedAvg with non-i.i.d. data.

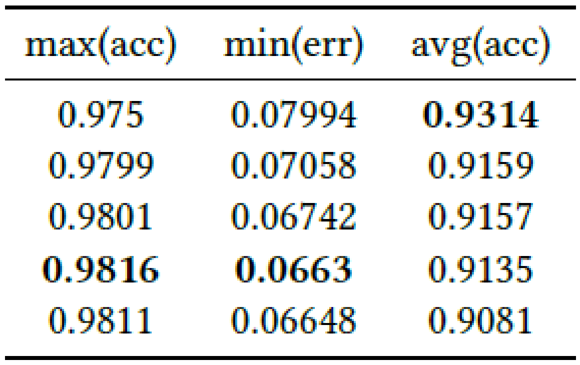
FedAvg min, max, and avg accuracy achieved, the best value is boldface:



max(acc)	min(err)	avg(acc)
0.9767	0.101	0.9582
0.978	0.109	0.96
0.9788	0.09794	0.9638
0.979	0.1325	0.9562
0.9792	**0.09315**	**0.9658**
0.6962	0.8643	0.09896

Figure 2.4: FedAvg results - paper 1

FSVRG min, max, and avg accuracy achieved, the best value is boldface:



max(acc)	min(err)	avg(acc)
0.975	0.07994	**0.9314**
0.9799	0.07058	0.9159
0.9801	0.06742	0.9157
0.9816	**0.0663**	0.9135
0.9811	0.06648	0.9081

Figure 2.5: FSVRG results - paper 1

In the paper [7]. Since Google's proposition and introduction of federated learning, it has opened largely interesting conceptual field that that many researchers and developers are trying to explore, as this new environment attracts more and more attention, and with many new algorithms being introduced into the federated learning field, it needs even more testing and clarification.

To that end, and in order to evaluate some of the promising algorithms being introduced, a series of experiments were being conducted, in this case the experiments were being done in the human activity recognition , which is said to fits the federated learning well because the patterns of the activities are usually generic on one hand while it still unique on the other hand, all the more, the collected data must also be private due to it sensitive nature, which is why human activity recognition fits well within the federated learning approaches.

Human activity recognition is based and depends on sensory devices, usually included within a smartphone or watches, identification and classification of human activities was separated as follows:

1. Physical activities: which includes climbing, jumping, walking and even sitting and standing.

2. Social activities: such as texting, talking, playing games, and watching.

Human activity recognition is particularly useful in health-care monitoring, especially for people with chronic diseases such as diabetes or high cholesterol diseases, it can also be used to monitor senior citizens who are in bad shape or in need for constant monitoring.

Physical activities are classified from recorded sensor data such as GPS or an audio.

In the dataset used which is REALWORLD dataset [8] there is a huge difference in the activity distribution amongst the classes, for example it states that in the "stairs" category, the activity represents 22% of the given data, however, the "jumping" activity only represents 2%, this imbalance should be addressed and considered when applying the learning approach.

In this paper the evaluation of the experiment was done to asses the performance of three promising algorithms which are FedAvg, FedPer and FedMa against a centralized training approach using a REALWOLD dataset [8] in which the data was gathered from 15 subject using 7 different devices and it consists of 8 different activities, the only pre-processing method that was done was a channel-wise z-normalization, the paper used a convolution neural network and compared its result against a traditional centralized training in federated learning.

Activities	**Instances**
Climbing Down	32047
Climbing up	37520
Jumping	6183
Lying	40843
Running	45581
Sitting	40747
Standing	40672
Walking	41555

Table 2.1: activity distribution

The subjects used in the dataset is treated as a client, for a total of 15 clients, as such each of the client's dataset is further separated into 80% - 20% ratio training and testing respectively, by combining the training and testing datasets, it generated a unified global training and testing datasets.

3 different evaluations were done on the 2 algorithms, FedAvg and FedMa. In the first evaluation, to see the results of the test as best as possible and to assess the performance differences between federated learning and centralized approach, an aggregation model was used on the server to test against the global dataset, because the model on the server itself has seen all the train datasets, although indirectly.

As for the second evaluation, by testing the model of each client's model with it own local dataset, it helps have a better understanding of the ability for each client to personalize.

As for the final evaluation, similar to the second evaluation the client's models were also used, and by testing it against the combined global datasets from all the clients, it was derived that the conclusion of this evaluation has given the very important understanding of how a client's model is able to perform with data that it has not seen, which only further shows one of federated learning most beneficial traits.

Seeing as how the FedPer algorithm does not have a global model, the evaluation was performed on only the two local, the one with where the client's model was testing again its own, and the one where it was testing again the combined datasets

Using the centralized training approach which was done on the same REALWORLD dataset with the use of Random Forest Classifier solution, and F-measure was calculated to be 81% [9]

Mostly all recent major solution incorporates an approach that is based on one form or another of convolutional neural network.

This study worked on two models , a two layered DNN model as well as the proposed CNN model which showed similar results, although both these two models (DNN/CNN) models were shown to have a better performance than the random forest classifier, with the CNN having even better result than the rest of them, as such the rest of the study focuses the results and findings of the CNN study rather than show both models

Literature	Models	F-measure
Original study	RFC	81%
This study	DNN	84.76%
This study	CNN	91.84%

Table 2.2: performance of classical centralized approaches on REALWORLD dataset

The accuracy of the CNN models was obtained from the global test dataset(91.84%), however using the same model individually on the local test datasets an accuracy that is higher was obtained, although it was not significantly higher than when testing on the global testing dataset, the mean accuracy score was 91.99%.

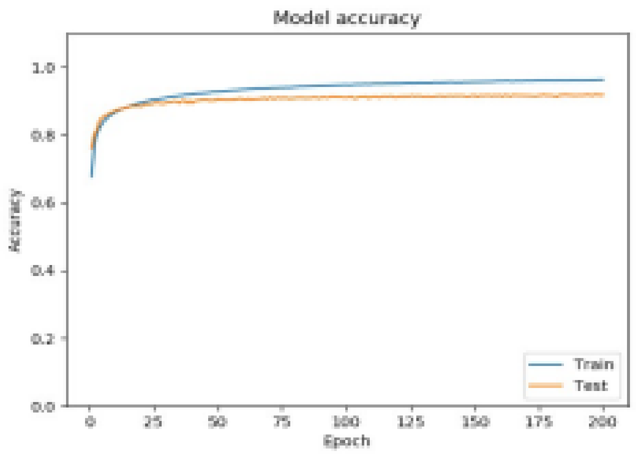


Figure 2.6: Model accuracy (centralized approach)

On a different note, in the centralized approach after handling each of the 15 client's data separately, using the CNN model on their own local dataset and without the use of any federated learning techniques, when tested on their own local test dataset, an accuracy of 95.41% was achieved, however when tested on the global test dataset, the accuracy dropped to 52.05%

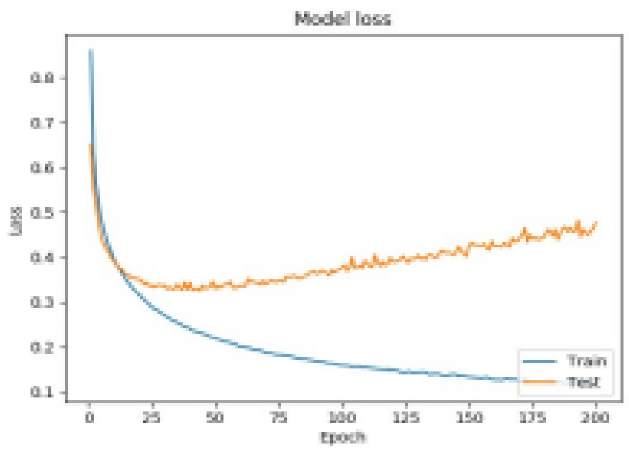


Figure 2.7: Model loss (centralized approach)

The FedAvg approach, in this approach, in order to give new weights, the server averages the weights of all the different local models, with this a new aggregation model, by applying the algorithm FedAvg with this modified convolutional neural network obtaining a model albeit on one hand it generated a performance of 82.74% accuracy which is much lower as opposed to the centralized approach which generated an accuracy of 91.84%, on the other hand at the end of each communication round it was able to receive a model which it could independently train with within their own datasets. It was noticed that when working on data it has not yet seen it was very beneficial on their performance as they achieved a 71.22% accuracy on the model of the combined global dataset, which is significantly higher when compared to the 52.05% of local learning model without federated learning techniques.

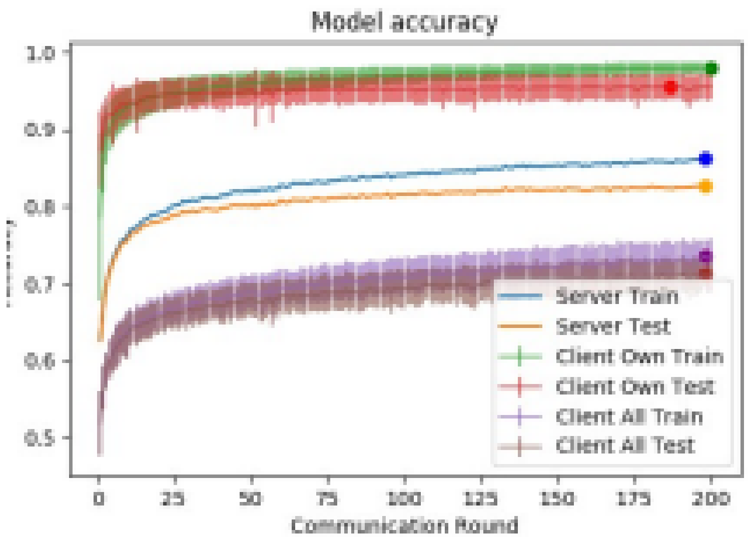


Figure 2.8: Model accuracy(FedAvg approach)

This apparently has no downside on the client's personalization abilities, the client's models when using FedAvg algorithm achieved an accuracy of 95.55% which is to say not a significant improvement from the standard centralized local learning approach. It was deduced that the more iterations the model slowly but surely improves, although the FedAvg approach requires large number of epochs than in the centralized approach.

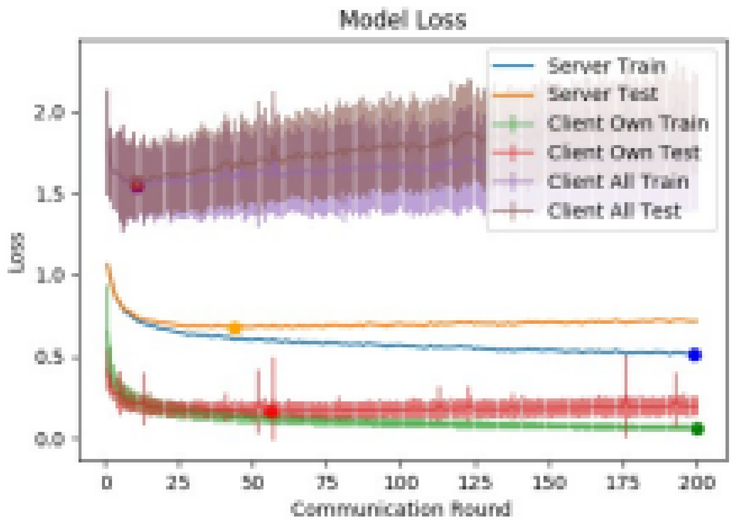


Figure 2.9: Model loss(FedAvg approach)

Federated personalization, this approach the model is separated into two layers, a base layer and a personalized layer, the base layer is the only layer that is aggregated to the server as such the personalized layer is not communicated. After testing the models, it was found that the client's are able keep their abilities to personalize and achieve a well-earned performance with an accuracy of 95.05% on the local dataset which is only slight less than the centralized local learning accuracy of 95.41%.

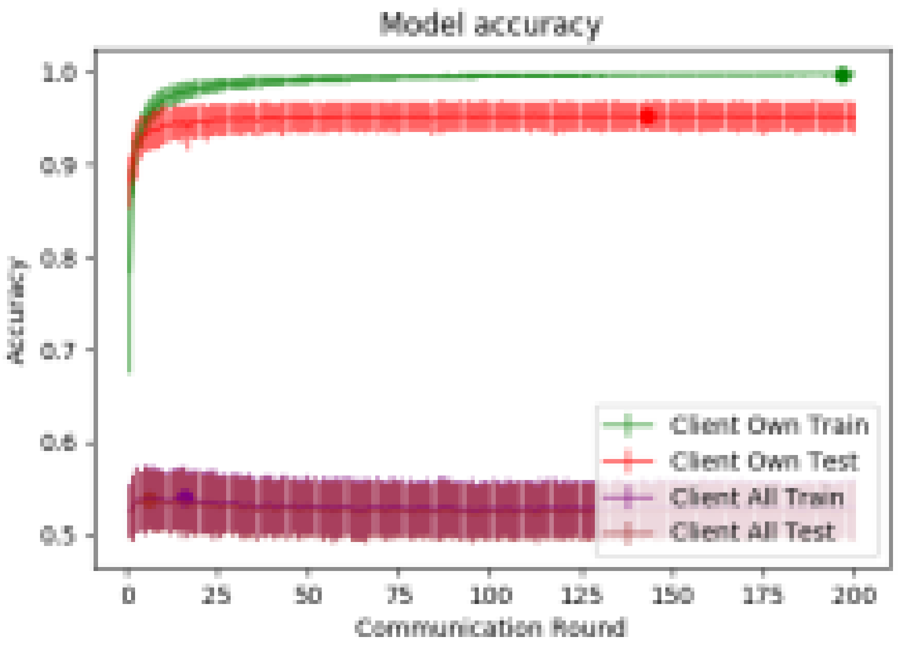


Figure 2.10: Model accuracy(FedPer approach)

However, on the global set it was able to achieve a slightly better accuracy of 52.51%

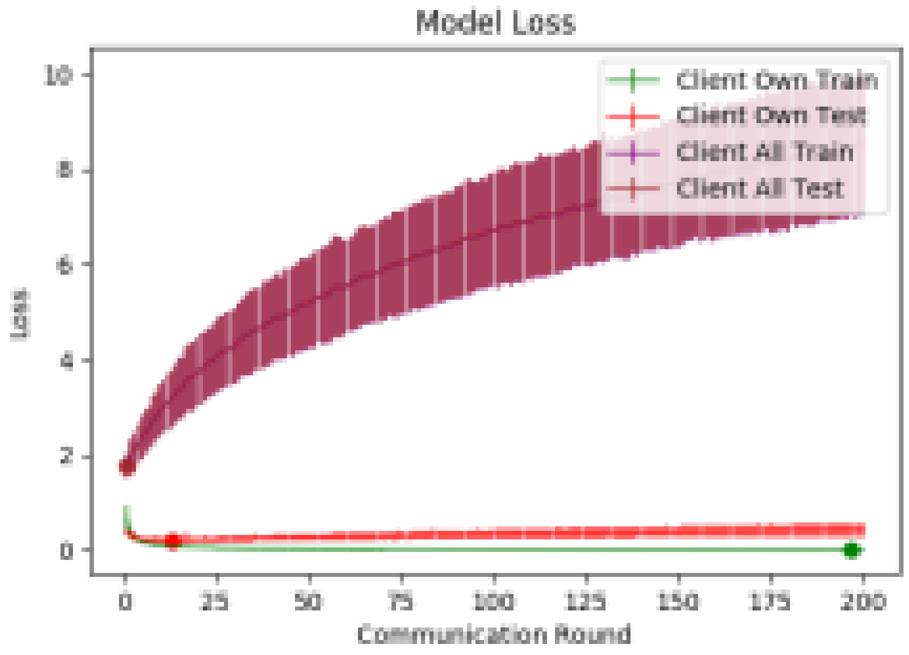


Figure 2.11: Model loss(FedPer approach)

Federated match averaging, which is similar to federated personalization in the sense that it is a layer-wise learning scheme, that inhabits the matching and merging of nodes with similar weights, layers are trained and communicated to the servers independently, two communication are required per communication round, one for transmitting layers weight and the other is for matching.

Using the FedMa with the two layered CNN, it got the accuracy of 77.91% on the server model which is far lower than the centralized approach and lowest of all approaches.

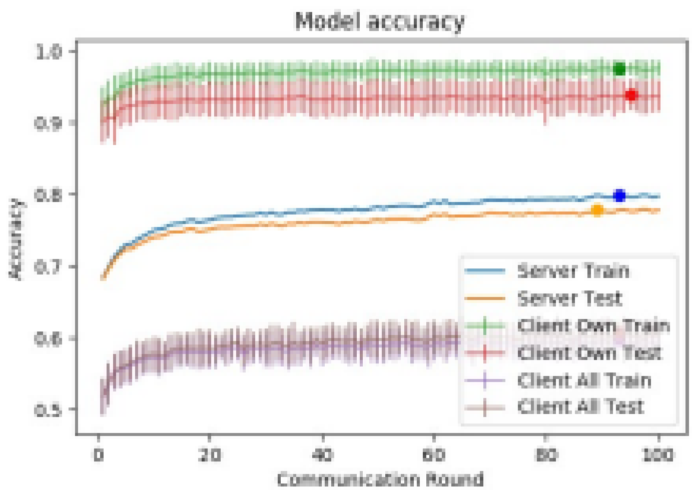


Figure 2.12: Model accuracy(FedMa approach)

The clients were also able to keep their ability to personalize with minor down sides, on the local test-set it was able to achieve an accuracy of 93.80% while on the global test-set the accuracy was measured at 60.77%, similar to FedAvg it showed regular growth as the training continued.

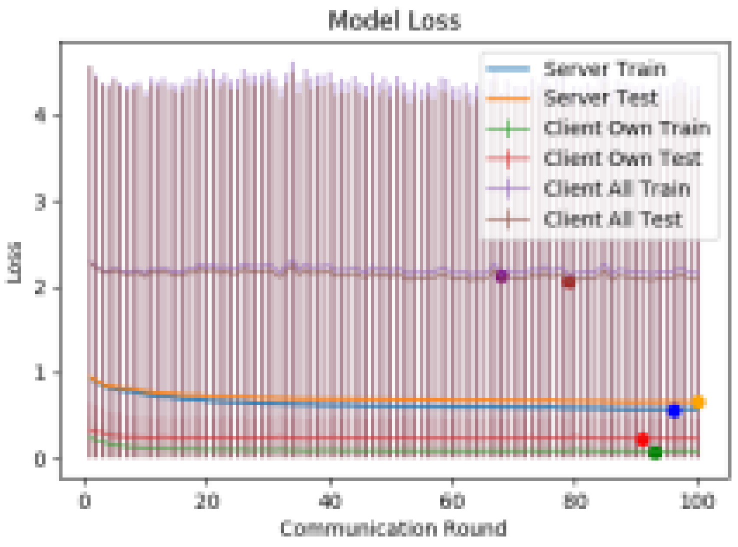


Figure 2.13: Model loss(FedMa approach)

In summery this table shows the difference in comparing between the centralized approach and federated learning algorithms

Approach	Server accuracy	F-Client own accuracy	All client accuracy
Centralized learning	91.84%	91.99%	N/A
Local Learning	N/A	95.41%	52.05%
FedAvg	82.74%	95.55%	71.22%
FedPer	N/A	95.05%	52.51%
FedMA	77.91%	93.80%	60.77%

Table 2.3: centralized and federated learning results (realworld dataset)

In the paper [10]. It's an attack in that the attacker can access the client and server communication to recover some clients' private data. And attack the federated learning scheme.

These attacks make some doubts about privacy in federated learning, The research found assumptions that can weaken the attacks.

The formula for recovering input from gradient inversion:

$$argmin_{x} L_{grad}(x;\theta,\nabla_\theta L_\theta(x^*,y^*)) + \alpha R_{aux}(x) \tag{2.1}$$

The Gradient:

$$\nabla_\theta L_\theta(x^*,y^*) \tag{2.2}$$

Regularizes the image based on the previous one:

$$\alpha R_{aux}(x) \tag{2.3}$$

Some ideas for defenses against Gradient Inversion Attack:

- Encrypt gradient.

- Perturbing gradient.

- Weak encryption of inputs.

The experiments show that if the size of the batch is larger than 32 it will weaken the attack if lower than 32 its will be not safe against the strongest attack. Federated learning should use a combining multiple defensive mechanisms.

2.3 Overview of implementation Tools

There are many open-source framework and software options available that implement federated learning, to choose the right platform for our research, we need first to have a brief idea of what each of these platforms offers and in what domain are they used.

2.3.1 TenserFlow Federated (TFF)

TensorFlow Federated (TFF) is a Python 3 open-source framework for federated learning developed by Google. The main motivation behind TFF was Google's need to implement mobile keyboard predictions and on-device search. TFF is actively used at Google to support customer needs.

2.3.2 NVIDIA Clara

A platform for AI applications and accelerated frameworks for healthcare researchers, developers, and medical device makers, creating AI solutions to improve healthcare delivery and accelerate drug discovery.

2.3.3 PySyft and PyGrid

PySyft is an open-source Python 3 based library that enables federated learning for research purposes and uses FL, differential privacy, and encrypted computations. It was developed by the OpenMined community and works mainly with deep learning frameworks such as PyTorch and TensorFlow. PyGrid implements federated learning on web, mobile, edge devices, and different types of terminals. PyGrid is the API to manage and deploy PySyft at scale. It can be controlled using PyGrid Admin.

2.3.4 Chosen Tool

As we discussed before, Google was the company that introduced Federated learning in the first place, so when choosing a tool to work with it only makes sense to use their own tool, TenserFlow Federated is a great tool to implement federated learning methods on, it's easy to use even for developers with no experience in machine learning, it has a good visualization, and has a great documentation and tutorial videos.

Chapter 3

Analysis

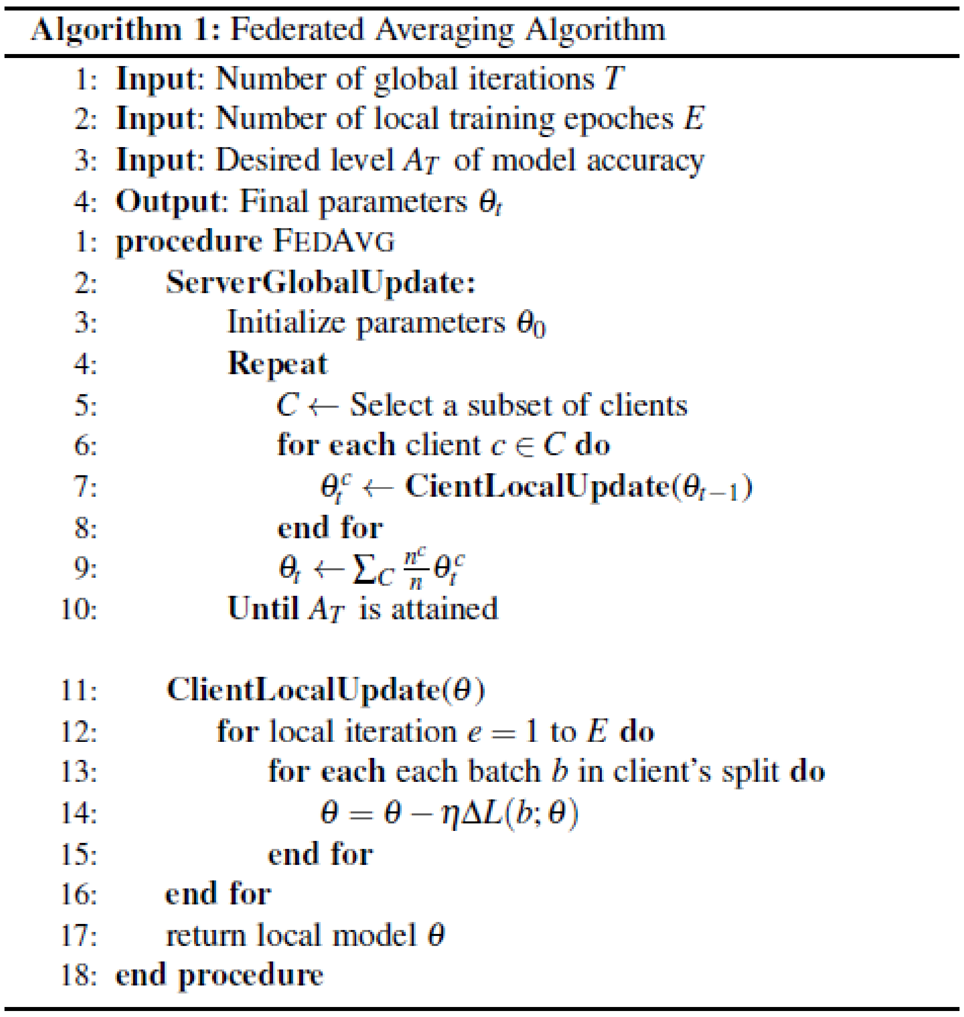
In this section we will be getting in more depth and analyzing each of the aggregation algorithms for the methods that we presented previously.

3.1 FedAvg algorithm

Below we describe the logic of the FedAvg algorithm.

- Step 1: The server initializes the parameter of the training model.
- Step 2: Choose the appropriate set of clients to participate in the training.
- Step 3: Each client receives the global model.
- Step 4: Each client trains the model on his set of training data.
- Step 5: Clients send the model back to the server.
- Step 6: The server minimizes the global loss function through averaging over the received gradients.

These steps keep on repeating until the global loss function converges, or until a desirable accuracy level is attained.



Algorithm 1: Federated Averaging Algorithm

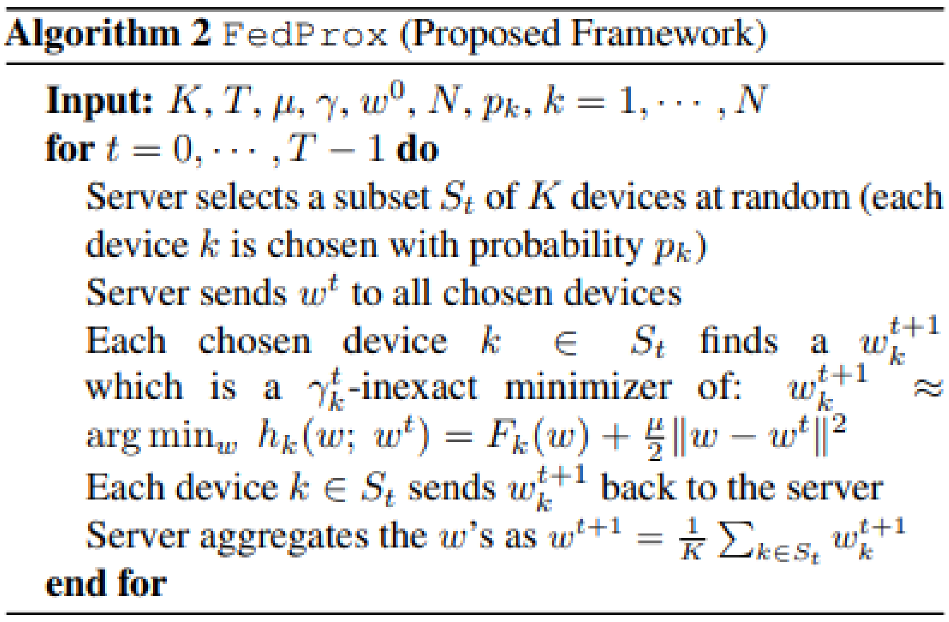
```
Input: Number of global iterations $T$
Input: Number of local training epoches $E$
Input: Desired level $A_T$ of model accuracy
Output: Final parameters $\theta_t$
procedure FEDAVG
    ServerGlobalUpdate:
        Initialize parameters $\theta_0$
        Repeat
            $C \leftarrow$ Select a subset of clients
            for each client $c \in C$ do
                $\theta_t^c \leftarrow$ CientLocalUpdate($\theta_{t-1}$)
            end for
            $\theta_t \leftarrow \sum_C \frac{n^c}{n} \theta_t^c$
        Until $A_T$ is attained

    ClientLocalUpdate($\theta$)
        for local iteration $e = 1$ to $E$ do
            for each each batch $b$ in client's split do
                $\theta = \theta - \eta \Delta L(b; \theta)$
            end for
    end for
    return local model $\theta$
end procedure
```

Figure 3.1: FedAvg Algorithm

3.2 FedProx algorithm

FedProx is designed to handle heterogeneous problems in federated learning. In federated learning, each participating device or user has its own unique data distribution and model parameters. This heterogeneity can lead to sub optimal performance of the global model if not properly addressed. FedProx addresses this issue by considering the proximity of the local user models to the global model during the aggregation process, allowing it to effectively incorporate the unique characteristics of each local user model into the global model. This results in a more robust global model that can handle heterogeneity in the data and user models.



```
Algorithm 2 FedProx (Proposed Framework)
Input: $K$, $T$, $\mu$, $\gamma$, $w^0$, $N$, $p_k$, $k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ do
    Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with probability $p_k$)
    Server sends $w^t$ to all chosen devices
    Each chosen device $k \in S_t$ finds a $w_k^{t+1}$ which is a $\gamma_k^t$-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
    Each device $k \in S_t$ sends $w_k^{t+1}$ back to the server
    Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for
```

Figure 3.2: FedProx Algorithm

Here is the algorithm for FedProx:

- Initialize the global model w with random values and set a learning rate lr.
- On each participating device or user, locally update the model w_i by running a number of training steps using the local data.
- Compute the average model w_bar from all participating devices or users: $w_bar = (sum_i w_i)/n$, where n is the number of participants.

- Compute the proximity d_i between the local model w_i and the average model w_bar : $d_i = ||w_i - w_bar||^2$.
- Compute the weighted average of the local models w_hat using the proximity values d_i as weights: $w_hat = sum_i(d_i * w_i)/sum_i d_i$.
- Update the global model w by performing a proximal gradient step with the weighted average model w_hat and the learning rate lr: $w = w - lr * (w_hat - w)$.
- Repeat steps 2 to 6 for a desired number of rounds or until the performance of the global model converges.

3.3 FedPer algorithm

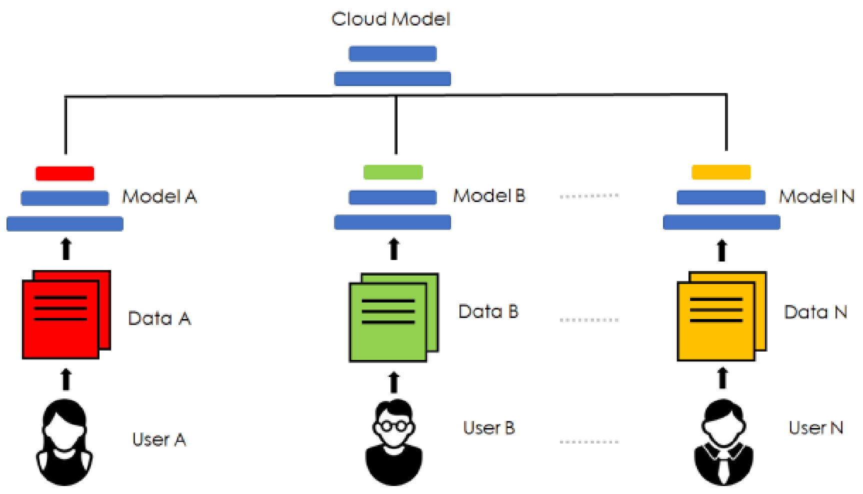


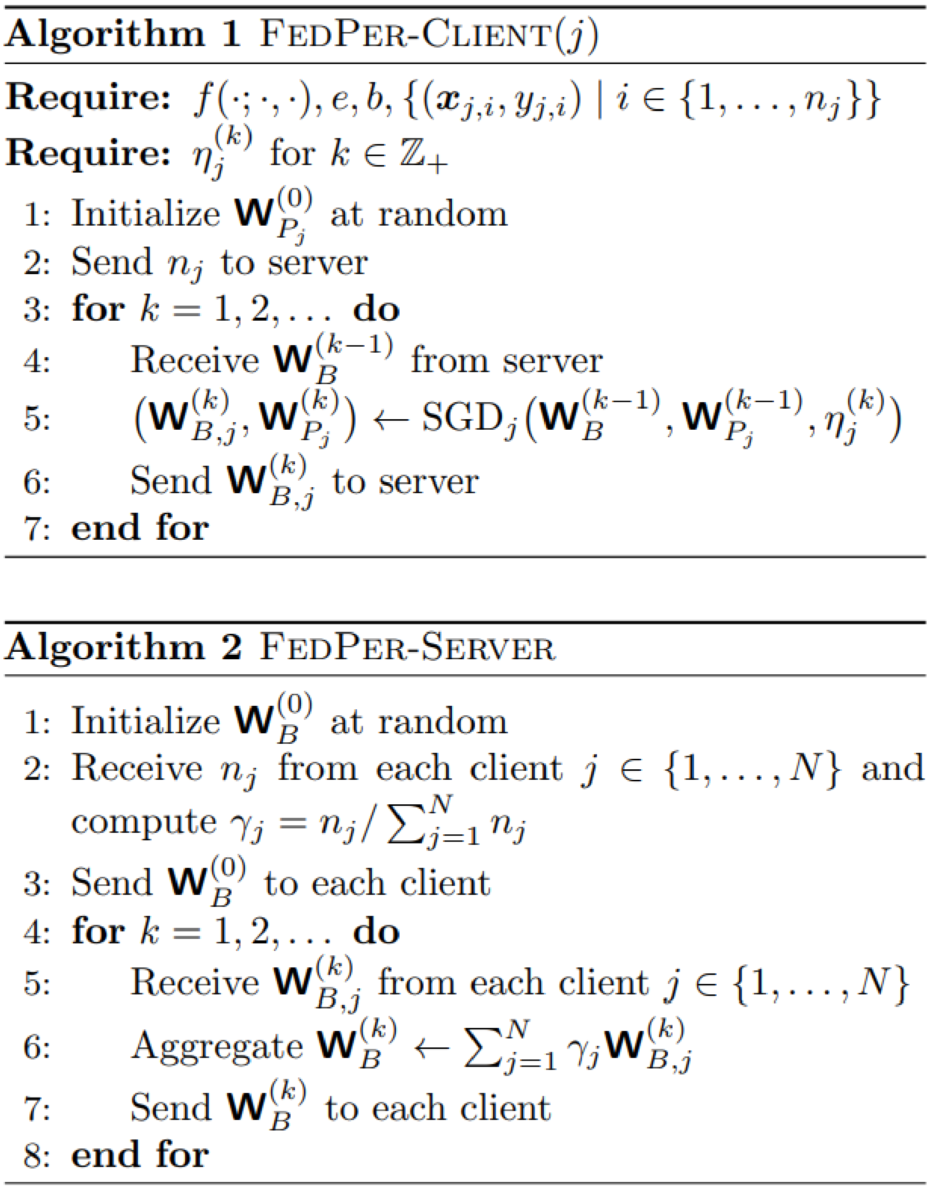
Figure 3.3: personlization and base layer

In the figure [3.3] it shows the proposed approach (personalization layer), in which they all have the same base layers which is shown by the color blue, the difference is the personalized layers which is different and unique, the personalized layer cannot be shared with a parameter server and is kept confidential , only the base layer is shared.

The research community has not as of yet uncovered the full possibilities of the personalized federated learning, and it still faces many problems each algorithms aims to solve some of these challenges which federated learning faces.

The main goal of the base and personalization approach is generalized assumptions which is what is proposed in FedPER, the base layers which are shared acquires knowledge or information from the people or client, and the personalization layers captures user specific information, which was proposed in a similar approach “multi-task and transfer learning”[11].

As shown in figure [3.3] where it was discussed that base layer are shared through the parameter server while the personalized layers are not, FedPER denoted that number of base and personalized layers on each client by positive integers KB and KP respectively and let there be N user devices. Let us designate the base layer weight matrices by WB,1,WB,2, . . ., WB,kb, and the proposed algorithms:



```
Algorithm 1 FedPer-Client($j$)
Require: $f(\cdot;\cdot,\cdot), e, b, \{(\boldsymbol{x}_{j,i}, y_{j,i}) \mid i \in \{1,\ldots,n_j\}\}$
Require: $\eta_j^{(k)}$ for $k \in \mathbb{Z}_+$
1: Initialize $\mathbf{W}_{P_j}^{(0)}$ at random
2: Send $n_j$ to server
3: for $k = 1, 2, \ldots$ do
4:     Receive $\mathbf{W}_B^{(k-1)}$ from server
5:     $(\mathbf{W}_{B,j}^{(k)}, \mathbf{W}_{P_j}^{(k)}) \leftarrow \mathrm{SGD}_j(\mathbf{W}_B^{(k-1)}, \mathbf{W}_{P_j}^{(k-1)}, \eta_j^{(k)})$
6:     Send $\mathbf{W}_{B,j}^{(k)}$ to server
7: end for
```

```
Algorithm 2 FedPer-Server
1: Initialize $\mathbf{W}_B^{(0)}$ at random
2: Receive $n_j$ from each client $j \in \{1,\ldots,N\}$ and
   compute $\gamma_j = n_j / \sum_{j=1}^N n_j$
3: Send $\mathbf{W}_B^{(0)}$ to each client
4: for $k = 1, 2, \ldots$ do
5:     Receive $\mathbf{W}_{B,j}^{(k)}$ from each client $j \in \{1,\ldots,N\}$
6:     Aggregate $\mathbf{W}_B^{(k)} \leftarrow \sum_{j=1}^N \gamma_j \mathbf{W}_{B,j}^{(k)}$
7:     Send $\mathbf{W}_B^{(k)}$ to each client
8: end for
```

Figure 3.4: FedPer algorithm

FedAVG's approach is to train what is considered a base layer globally whereas each client updates its base and personalized layers locally.

Evaluation was done on the performance of FedPER using many datasets and deep-learning models in comparison with FedAVG, it was noticed the FedPER was reduced to or resembled FedAVG when the personalization layer was removed.

It was theorized that the base layer is redundant and that the personalized layer was enough, to test this a fully connected linear layer replaced the base layer, if it did not have a huge gap in accuracy then it would've been true

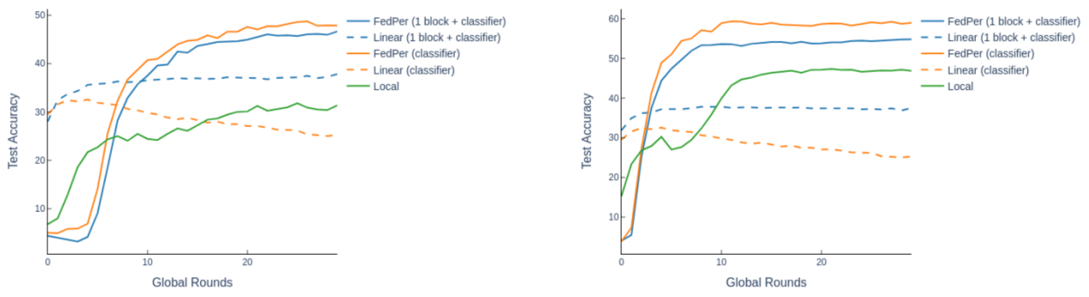


Figure 3.5: linear layers accuracy

These figures [3.5] show that the base layer is in fact not true in this case [12].

3.4 FedSGD algorithm

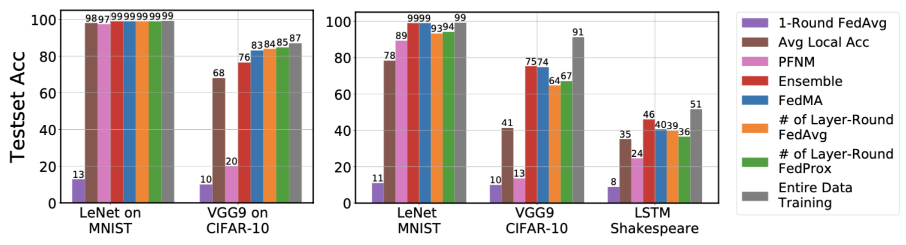
Formula for FedSGD:

$$F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \tag{3.1}$$

$$g_k = \nabla F_k(w_t) \tag{3.2}$$

3.5 FedMA algorithm

This figure [3.6] is comparison among federated learning algorithms, between two partitions: homogeneous data and heterogeneous data, using LeNet trained on MNIST, VGG9 trained on CIFAR-10, and LSTM trained on Shakespeare dataset. Based on the figure [3.6], in the heterogeneous data partition FedMa outperforms FedAvg, but fails on deep architectures necessary to solve complex tasks.



(a) Homogeneous (b) Heterogeneous

Figure 3.6: FedMa test accuracy

To increase the performance of FedMa it should do communication, local clients receive the matched global model and reconstruct their local models with the size equal to the original local model.

Formula for Federated Matched Averaging of neural networks:

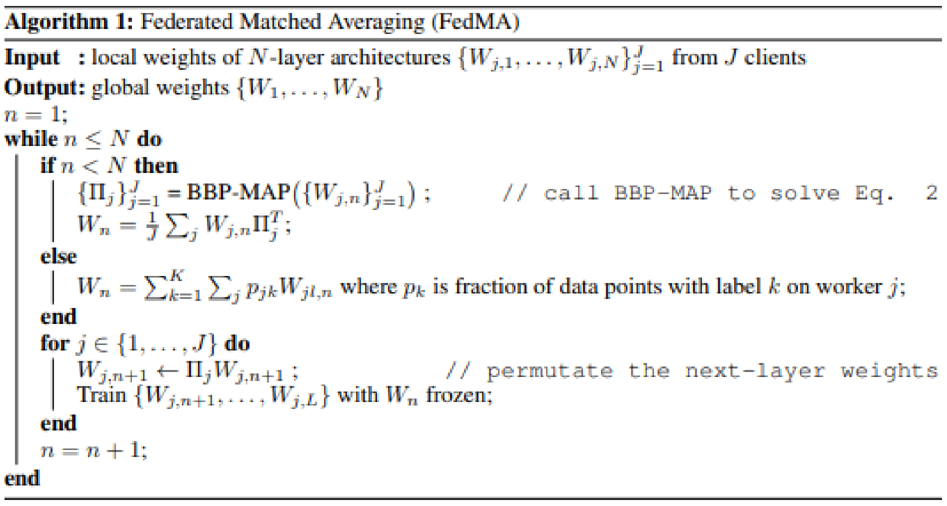
Fully connected architecture:

$$\hat{y} = \sigma(_x W_1 II) II^T W_2 \tag{3.3}$$

- Where II is any L x L permutation matrix.

- (o) = non-linearity.

- (W) = Weight.



Algorithm 1: Federated Matched Averaging (FedMA)

```
Input  : local weights of $N$-layer architectures $\{W_{j,1}, \ldots, W_{j,N}\}_{j=1}^{J}$ from $J$ clients
Output: global weights $\{W_1, \ldots, W_N\}$
$n = 1$;
while $n \leq N$ do
    if $n < N$ then
        $\{\Pi_j\}_{j=1}^{J}$ = BBP-MAP$(\{W_{j,n}\}_{j=1}^{J})$ ;        // call BBP-MAP to solve Eq.  2
        $W_n = \frac{1}{J}\sum_j W_{j,n}\Pi_j^T$;
    else
        $W_n = \sum_{k=1}^{K}\sum_j p_{jk}W_{jl,n}$ where $p_k$ is fraction of data points with label $k$ on worker $j$;
    end
    for $j \in \{1, \ldots, J\}$ do
        $W_{j,n+1} \leftarrow \Pi_j W_{j,n+1}$ ;              // permutate the next-layer weights
        Train $\{W_{j,n+1}, \ldots, W_{j,L}\}$ with $W_n$ frozen;
    end
    $n = n + 1$;
end
```

Figure 3.7: FedMa algorithm

3.6 FedPAQ algorithm

FedPAQ was planned to face these challenges using periodic averaging, partial device participation and last but not least quantized message passing.

1. Periodic averaging: instead of letting participant nodes communicate with a parameter server after each round of its training, thus resulting in a backing up or slowing down of the network, FedPAQ proposes to use periodic averaging which lets the participant nodes multiple updates and iterations of its training before synchronizing with the parameter server, meaning each time a node takes an updated model from the server, it runs t iterations before sending its information to the server for updating, thus reducing the round of communication between the server and nodes, and therefore reducing the communication cost of training. In FedPAQ the plan is to run T iterations at every node and nodes need to communicate with the server $K = T/t$ rounds, thus reducing the communication by a factor of $1/t$. If the goal is to procure a more optimal and precise accuracy, then it is not necessarily optimal to choose a very large number for the value of t, and by doing so the noise of the system increases and the local models are instead optimized. Therefore, it might be better to run more iterations of T rather than t to achieve a better result and accuracy.

2. Partial node participation: usually in a federated learning network a large amount of devices contributes to the communication which is done through a base station, and there is a limitation on the download bandwidth of a base station, therefore, not all devices are able to upload their message at the same time, thus decreasing the speed of the training process severely. In addition, should all devices participate in the training process this will cause a huge bottle-neck on the network, which would cost more, In reality, there are many agents that contribute to whether a device is can participate in the training, for example if a device is out of range to the base station, if it is idle or not, or if it is charging, therefore, some of the devices may be considered inactive.

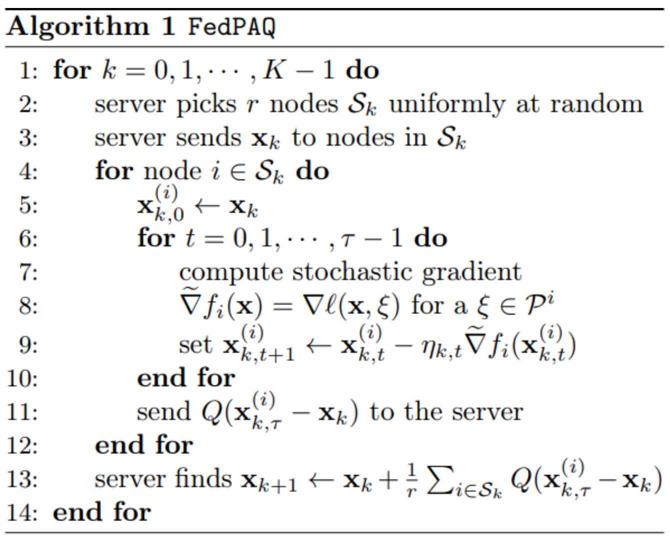
FedPAQ methods captured the limitations mentioned before, assuming that among the total of n devices, only r nodes ($r <= n$) are available in each round of the training. It is also assumed that due to the availability criteria that was mentioned before, active devices are distributed randomly and uniformly over the network, such that in every training period k=0,1, . . . , K − 1 of the algorithms, the parameter server sends the current model Xk to all the r nodes in subset Sk are distributed randomly and uniformly among n nodes[13].

3. Quantized message-passing: quantization algorithms solves or takes part in solving another federated learning challenge which is communication bottle-neck, in this case due to the limited uplink bandwidth in which the devices communicates and updates models and parameters to the parameter server, it is necessary to reduce the size of the uploaded messages from the devices, thus FedPAQ proposed to apply an quantization on the messages sent from the devices to code them, thus causing a reduction in their size, Depending on the accuracy of the quantizer, the network communication overhead is reduced by exchanging the quantized updates. In the proposed FedPAQ, "each node obtains a model after running t local iterations of an optimization method on the most recent model which it has received form the server. Then each node applies a quantizer operator Q(·) on the difference between the received model and its updated model and uploads the quantized vector to the parameter server. Once these quantized vectors are sent to the server, it decodes the quantized signals and combines them to come up with a new model".

Using the previously mentioned algorithms and techniques FedPAQ was presented, the proposed method involved K periods, during which every node performs t local updates which brings about a total number of T = K t iterations, each period k = 0, · · · , K-1, the parameter server then picks r <= n nodes uniformly at random which they denoted by Sk, The parameter server then broadcasts its current model ×k to all the nodes in Sk and each node i E Sk performs t local Stochastic gradient descent (SGD) updates using its local dataset.

Results and discussion:

FedPAQ algorithm on one hand managed to tackle the communication bottle-neck problem and reduced the communication load with the use of the three methods or modules mentioned previously, on the other hand it reduced the convergence accuracy, therefore it needed more iterations in its training period. To try and evaluate FedPAQ they amounted the training time as the cost objective, they considered the total training time to be composed of communication and computation time," Consider T iterations of training with FedPAQ that consists of K = T /t rounds of communication. In each round, r workers compute t iterations



Algorithm 1 FedPAQ

```
for $k = 0, 1, \cdots, K-1$ do
    server picks $r$ nodes $\mathcal{S}_k$ uniformly at random
    server sends $\mathbf{x}_k$ to nodes in $\mathcal{S}_k$
    for node $i \in \mathcal{S}_k$ do
        $\mathbf{x}_{k,0}^{(i)} \leftarrow \mathbf{x}_k$
        for $t = 0, 1, \cdots, \tau - 1$ do
            compute stochastic gradient
            $\widetilde{\nabla} f_i(\mathbf{x}) = \nabla \ell(\mathbf{x}, \xi)$ for a $\xi \in \mathcal{P}^i$
            set $\mathbf{x}_{k,t+1}^{(i)} \leftarrow \mathbf{x}_{k,t}^{(i)} - \eta_{k,t} \widetilde{\nabla} f_i(\mathbf{x}_{k,t}^{(i)})$
        end for
        send $Q(\mathbf{x}_{k,\tau}^{(i)} - \mathbf{x}_k)$ to the server
    end for
    server finds $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q(\mathbf{x}_{k,\tau}^{(i)} - \mathbf{x}_k)$
end for
```

Figure 3.8: FedPAQ Algorithm

of SGD with batch size B and send a quantized vector of size p to the server”

Communication time: The total number of uploaded bits divided by bandwidth (BW) in each round is defined as communication time, r · |Q(p, s)| is the total number of bits in each round, whereas |Q(p, s)| is the number of bits required to encode a quantized vector of dimension p, Assuming pF bits is used to represent an unquantized vector of length p, where F is typically 32 bits.

Computation time: following a wildly shifted exponential model for gradient computation time[14].

Computing the gradients in a period with t iterations and using batch size B takes a deterministic shift t · B · shift as well as a random exponential time with mean value (t · B · (scale)to power of -1), defining the communication/computation ratio as

$$\frac{C_{comm}}{C_{comp}} = \frac{pF/BW}{shift + 1/scale} \tag{3.4}$$

In all of the experiments done a batch size B = 10 was used. [15]

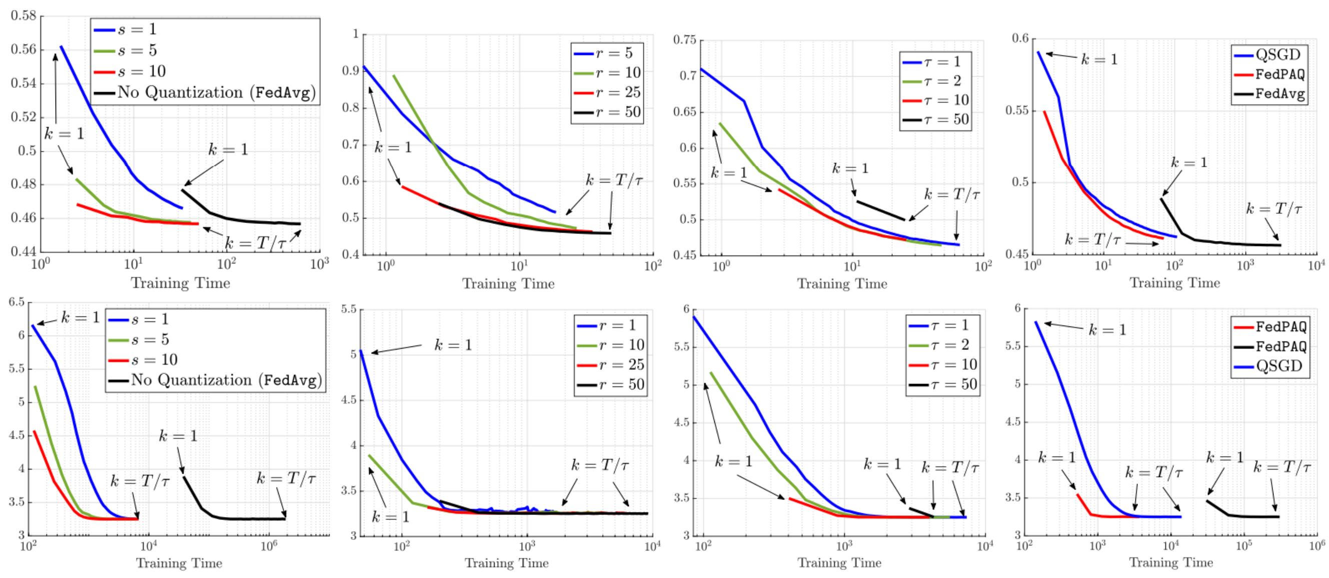


Figure 1: Training Loss vs. Training Time: Logistic Regression on MNIST (top). Neural Network on CIFAR-10 (bottom).

Figure 3.9: training loss vs training time

3.7 Proposed Method

In this section, we introduce our proposed method in federated learning. Our focus is on a scenario with multiple clients, a central server, and the goal of addressing a multi-class classification problem. We'll outline the key features and elements of this method, emphasizing its approach to asynchronous training, compress and decompress of data, and parameter aggregation to achieve optimized global models. Additionally, we'll discuss the model's communication efficiency and testing procedure to evaluate its performance in real-world applications.

3.7.1 Preliminaries

This study explores a federated learning scenario involving a collection C of N clients and a global federated server, where $C = \{c_1, \dots, c_N\}$. The focus is on a multi-class classification problem with a set Q containing c classes, denoted as $Q = \{1, \dots, c\}$. Each client c_i possesses a local dataset D_i with n_i instances and a set of classes from Q, where $1 \le i \le N$. Instances from D_i are represented as (x_{ij}, y_{ij}), with $1 \le j \le n_i$. When training the model on client c_i, the aim is to learn the association between x_{ij} and y_{ij} for all instances, $\forall j \in \{1 \le j \le n_i\}$, to construct a classifier Π_i. This classifier predicts class labels for unknown instances during testing. The batch size used for training on client c_i is B_i, and the number of Stochastic Gradient Descent (SGD) operations in one training round on c_i is denoted as τ_i. The estimation of τ_i is given by $\tau_i = \left\lfloor \frac{E \cdot n_i}{B_i} \right\rfloor$, where E represents the number of local epochs needed for training on c_i.

3.7.2 Mathematical Model

In this section, we propose a mathematical model for the asynchronous federated learning approach presented in the code. The objective of the model is to optimize a global model G based on the local models $L_1, L_2, ..., L_n$ trained on different clients' data. The optimization process is performed over multiple rounds, where in each round, the local models are trained asynchronously, and their parameters are then averaged to update the global model. The goal is to minimize the global loss function $F(G)$, which is defined as the average of the local loss functions $f_i(L_i)$ over all clients i:

$$F(G) = \frac{1}{n} \sum_{i=1}^{n} f_i(L_i) \tag{3.5}$$

Each local loss function $f_i(L_i)$ is defined as the cross-entropy loss between the predicted and true labels for the i-th client's data. The local model L_i is trained on the i-th client's data, which is denoted by D_i. The training process of L_i is performed using stochastic gradient descent (SGD) with a learning rate of α. In each round, the local models are trained for T epochs.

During the training process, each client's data D_i is divided into batches of size B. The local model L_i is trained on each batch j using the following update rule:

$$L_i^{j+1} = L_i^j - \alpha \nabla f_i(L_i^j), \tag{3.6}$$

where $\nabla f_i(L_i^j)$ is the gradient of the local loss function $f_i(L_i^j)$ with respect to the parameters of the local model L_i^j.

After the local models have been trained for T epochs, their parameters are sent to the server, where they are averaged to update the global model G. The average of the local models is performed using the following update rule:

$$G^{k+1} = \frac{1}{n}\sum_{i=1}^{n} L_i^k, \tag{3.7}$$

where k denotes the current round.

To improve the communication efficiency between the clients and the server, the local model parameters are compressed before being sent to the server. This is achieved by using a compression ratio r. The compressed parameters are then decompressed at the server using the following functions:

$$compress(v, r) = v[:: r] \tag{3.8}$$

$$decompress(v', n, r) = [0]_n; v'[:: r] = v \tag{3.9}$$

where v is the original parameter vector, v' is the compressed parameter vector, n is the size of the original parameter vector, and $[0]_n$ denotes a vector of size n with all elements set to zero.

In each round, the global model is tested on a separate validation dataset to evaluate its performance. The accuracy of the global model is measured as the percentage of correctly classified samples in the validation dataset.

3.8 Pros and cons of algorithms

Method	**Pros**	**Cons**
FedAvg	Performing more local epochs allows for more local computation and potentially reduced communication, which can greatly improve the overall convergence speed in communication constrained networks	With dissimilar local objectives, a larger number of local epochs may lead each device towards the optima of its local objectives as opposed to the global objective-potentially hurting convergence or even causing the method to diverge
FedProx	Solve heterogeneous problems	
FedPer	Base layer shared through the parameter server while the personalized layers are no	
FedSGD	Great in deep learning	Requires many rounds of training to produce great models
FedMA	Outperforms FedAvg	Fail on deep architecture necessary to solve complex tasks
FedPAQ	Managed to reduce the communication load	
Proposed Approach	Asynchronously train clients Compress and decompress the data	

Table 3.1: Pros and cons of algorithms

3.9 Dataset

In machine learning, datasets are used to train models, validate their performance, and make predictions on new data. The quality and diversity of the dataset are crucial factors in the success of the machine learning models, as the models can only learn from the data they are trained on.

To have a more clear result of the aggregation methods we have to use different types of datasets, in this section we will introduce the two datasets that will be used in our project.

3.9.1 EMNIST dataset

EMNIST dataset[16] consists of 671,585 images of digits and upper and lower case English characters (62 classes). The federated version splits the dataset into 3,400 unbalanced clients indexed by the original writer of the digits/characters. The non-IID distribution comes from the unique writing style of each person. the dataset was used by number of papers including [17] and [18].

3.9.2 MNIST dataset

A widely used dataset in the field of machine learning and computer vision. It stands for the "Modified National Institute of Standards and Technology" dataset. MNIST is a collection of handwritten digits, commonly used for training and testing various machine learning and deep learning algorithms for the task of image classification. The dataset contains 60,000 training images and 10,000 test images, each image is a 28x28 pixel gray-scale image of handwritten digit. Each image represents a single digit from 0 to 9. the dataset was used by number of papers including [1].

3.9.3 CIFAR-10

The CIFAR-10 dataset is a widely used and well-known dataset that consists of 60,000 color images, each of which is 32x32 pixels in size. These images are divided into ten classes, with each class representing a distinct object or category. The dataset includes the following classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. the dataset was used by number of papers including [4].

Chapter 4

Benchmark design

In this chapter we will discuss the proposed approach, and the different parameters it has. Introducing the different datasets that will be used, and the way they will be distributed among clients.

4.1 Exploring key parameters

In this section we will explore different key parameters. We have done multiple analysis and studies on the performance of well-known aggregation algorithms, as well as several parameters that are going to be the core of implementing and assessing the aggregation of each method, each of the parameters will have a different value.

To overcome the essential federated learning challenges, we study the following parameters: number of clients, number of communication rounds, number of epochs, amounts of data, learning rate, and number of devices. Finally, we will analyze the trade-offs between accuracy and convergence as well as the trade-off between redundancy and communication costs.

4.2 Data distribution

IID (independently and identically distributed) and non-IID (non-independently and identically distributed) data refer to the statistical properties of data samples and their differences.

1. IID Data (Independently and Identically Distributed): In the context of IID data, each data point in a dataset is drawn independently from the same probability distribution. This means that the probability distribution governing each data point is the same, and the data points are not influenced by or dependent on each other. Each data point is considered a random and unbiased sample from the population.

2. Non-IID Data (Non-Independently and Non-Identically Distributed): Non-IID data, on the other hand, means that the data points do not follow the same probability distribution, and they may exhibit dependencies or differences among them. Non-IID data is more complex and realistic, as it better represents real-world scenarios where data is often collected from various sources, devices, or populations, each with its own unique characteristics.

Once the dataset has been partitioned, each client can be assigned its own subset of the data, and the clients can use their data to train the model locally and send the updated model parameters to the server.

Chapter 5

Implementation

5.1 Tools

5.1.1 Python

Python is a high-level programming language that is widely used in various fields such as web development, scientific computing, data analysis, and artificial intelligence. It was first released in 1991 and has since become one of the most popular programming languages in the world due to its simplicity, flexibility, and powerful libraries. Python's syntax is easy to learn and read, making it an ideal language for beginners, while its vast collection of libraries and frameworks, such as NumPy, Pandas, and Scikit-learn, make it a powerful tool for data analysis and machine learning.

Python is particularly well-suited for machine learning due to its ease of use and extensive libraries. Machine learning is a subset of artificial intelligence that involves the use of algorithms to enable machines to learn from data and make predictions or decisions without explicit programming. Python's libraries such as NumPy, Pandas, and Scikit-learn provide a wide range of tools for data manipulation, preprocessing, visualization, and modeling, making it easy for developers to build and train machine learning models. Additionally, Python's simplicity and readability make it easy for non-experts to understand and modify the code, which is important for collaboration and knowledge sharing in the field of machine learning.

5.1.2 PyTorch

PyTorch is an open-source machine learning library based on the Python programming language. It was developed by Facebook's AI Research team to provide a flexible and efficient tool for building and training deep neural networks. PyTorch is designed to provide a smooth and intuitive user experience, making it easy for researchers and developers to experiment with different ideas and models.

PyTorch's key features include dynamic computation graphs, which allow for flexible and efficient model building, as well as a comprehensive set of tools for data loading, transformation, and visualization. It also provides a powerful debugging and profiling interface, which makes it easy to identify and diagnose issues with complex models. PyTorch has gained widespread adoption in the machine learning community due to its ease of use, flexibility, and scalability, and is used in a wide range of applications, from natural language processing and computer vision to robotics and autonomous systems.

5.1.3 Google Collaboration

Google Collaboration is a powerful suite of cloud-based tools that fosters teamwork, creativity, and productivity among its users. Offering a range of applications such as Google Docs, Sheets, Slides, and Drive, this platform enables teams to work together seamlessly on documents, spreadsheets, and presentations in real-time. Its intuitive interface and advanced sharing capabilities make it a go-to solution for businesses, educational institutions, and individual users alike. With features like live co-editing, version control, and commenting, Google Collaboration breaks down geographical barriers and streamlines communication, empowering teams to efficiently accomplish their goals and drive innovation.

5.1.4 Github

GitHub is a web-based platform that provides a collaborative environment for developers to store, manage, and share their code. It is built on top of the Git version control system, which allows developers to track changes to their code over time, collaborate with others, and merge changes seamlessly. GitHub offers a wide range of features, including issue tracking, project management tools, code reviews, and continuous integration and deployment. It also provides a platform for open-source projects, allowing developers to contribute to and collaborate on projects with others around the world. GitHub has become a crucial tool for software development teams, enabling them to work more efficiently, improve code quality, and accelerate the development process.

5.2 Parameters

5.2.1 Number of clients

The number of clients in federated learning can have a significant impact on the overall performance and accuracy of the model. More clients generally lead to better performance and increased accuracy due to a larger and more diverse dataset. However, it is important to note that increasing the number of clients can also lead to challenges in model convergence and communication overhead. Therefore, it is important to strike a balance between the number of clients and the computational resources available for training. Additionally, analyzing the performance of individual clients in federated learning can provide valuable insights into areas for improvement. By analyzing the performance of individual clients in federated learning, organizations can identify clients with high-quality data and use this information to improve model accuracy.

In studies involving CIFAR-10 and Fashion MNIST datasets, the number of clients in federated learning was found to have a significant impact on model performance. Increasing the number of clients led to improved performance due to a larger and more diverse dataset[19].

5.2.2 Batch size

Batch size in federated learning refers to the number of training examples used for each iteration of stochastic gradient descent on a client device. This allows the client to update its local model with less computational overhead. Additionally, the batch size can significantly affect the quality of the learned model in federated learning. By partitioning the local dataset into smaller batches for training on each client, federated learning can achieve more efficient computation and communication while maintaining data privacy.

The choice of batch size in federated learning can have a significant impact on the model's performance. Larger batch sizes can lead to faster convergence, but may also result in less accurate models due to overfitting. On the other hand, smaller batch sizes can offer greater accuracy, but at the expense of slower convergence and higher computational overhead[20].

5.2.3 Epochs

Epochs in Federated Learning refer to the number of times that a local client goes through its entire data set during each round of training. The local epochs can greatly impact the results of Federated Learning. The number of local epochs can affect the overall quality and performance of a model during Federated Learning.

The number of local epochs can impact the model performance in Federated Learning in several ways. On one hand, increasing the number of local epochs can improve the model's accuracy by allowing for more iterations and adjustments to be made on the local client data. However, an increased number of epochs can also lead to overfitting of the model, which negatively impacts its performance[21].

5.2.4 Learning rate

In federated learning, the learning rate determines how quickly the model should adapt to new data while training and updating its parameters at every iteration.

The choice of the learning rate has a significant impact on the outcomes of federated learning models. If the learning rate is too high, the model may overshoot and fail to converge to optimal values, leading to poor accuracy. Conversely, if the learning rate is too low, the model may converge slowly or may not converge at all within an acceptable number of iterations. The optimal value of the learning rate will depend on various factors such as the dataset used, network architecture, and optimization algorithm[22].

5.2.5 Compression ratio

Compression ratio in federated learning refers to the reduction of data size during communication between client devices and the central server without sacrificing model accuracy.

The compression ratio has a significant impact on the model performance in federated learning. While compression techniques effectively decrease communication costs, the overall model performance can suffer due to reduced model information during the compression-decompression process. Therefore, it is crucial to find the right balance between communication costs and model accuracy by optimizing the trade-off between them[23].

5.3 Experiments

We compared our proposed approach with a known algorithm (FedAvg) as a baseline, as we plan on comparing several other known algorithms in the future. We conducted an experiment on the MNIST dataset that consists of handwritten digits, it has a training set of 60,000 examples, and a test set of 10,000 examples. We used Neural networks as a model due to its ability to automatically learn hierarchical features from raw data. The complex and non-linear relationships within images are effectively captured by the intricate layers of a neural network, enabling it to discern patterns and nuances essential for accurate classification. We used PyTorch to implement the baseline as well as our approach, we use a compression and decompression method when updating the model, we also train the clients model Asynchronously, both these methods are used to reduce communication bottleneck and increasing communication efficiency, we used an optimizer setting the learning rate at 0.1 initially. The batch size at 32. The number of clients at 12. The number epochs at 20, and a compression ratio of 9. We slowly change the parameters incrementally to compare between the two approaches as the parameters differ.

To figure out the best set of parameters we had to run the code number of times each time with a different value to find out the best results, next we will show each parameter in a table with the number of testing values and results.

These are the default parameters:

Parameter	**Value**
number of clients	10
Batch size	32
Epochs	2
Learning rate	0.01
compression rate	5

Table 5.1: default parameters

The number of clients parameter:

Num_clients	**FedAvg**		**Proposed Approach**	
	Loss	**Accuracy**	**Loss**	**Accuracy**
10	1.3444	52.86%	0.9978	74.08%
12	**1.3100**	**58.53%**	**1.0103**	**76.53%**
15	1.8051	41.10%	1.3949	71.27%
16	1.9768	19.23%	1.6362	64.23%
20	2.0777	24.33%	1.8477	48.21%

Table 5.2: Number of clients parameter

The epochs parameter:

Epochs	FedAvg		Proposed Approach	
	Loss	Accuracy	Loss	Accuracy
1	2.0524	36.03%	1.8408	56.74%
2	1.3444	52.86%	0.9978	74.08%
3	1.4295	52.22%	0.6313	81.54%
4	0.7818	72.71%	0.5139	85.79%
5	0.7684	74.16%	0.4475	87.52%
6	0.8186	72.92%	0.4142	88.00%
7	0.7483	76.71%	0.3874	88.56%
8	0.5852	81.08%	0.3678	88.98%
9	0.8851	71.12%	0.3687	88.87%
10	0.6587	76.77%	0.3724	89.05%
11	0.5338	81.23%	0.3568	89.73%
20	**0.2471**	**92.58%**	**0.2292**	**94.28%**

Table 5.3: Epochs parameter

The batch size parameter:

Batch_size	**FedAvg**		**Proposed Approach**	
	Loss	**Accuracy**	**Loss**	**Accuracy**
30	1.255	61.55%	0.9253	77.32%
32	1.2581	57.44%	0.9952	74.32%
34	1.4605	51.87%	0.9712	75.53%
36	**1.3375**	**62.80%**	**1.0518**	**78.71%**
38	1.5267	46.58%	1.1386	72.69%
40	1.7934	38.39%	1.133	71.77%
42	1.4711	53.96%	1.218	72.88%
44	1.7256	44.39%	1.3962	72.69%
46	1.8451	34.61%	1.42	71.06%
48	1.7119	40.69%	1.3842	68.85%
50	1.7694	44.38%	1.5015	67.84%

Table 5.4: Batch size parameter

The learning rate parameter:

Learning_rate	**FedAvg**		**Proposed Approach**	
	Loss	**Accuracy**	**Loss**	**Accuracy**
0.01	1.1948	56.13%	0.8545	78.23%
0.02	0.7495	77.22%	0.5815	81.14%
0.03	0.8822	72.70%	0.617	80.58%
0.04	0.6108	80.79%	0.437	86.60%
0.05	0.589	81.49%	0.5226	82.14%
0.06	0.5565	83.43%	0.3947	87.77%
0.07	0.5445	82.30%	0.4247	86.50%
0.08	0.4481	85.89%	0.4481	86.59%
0.09	0.5521	80.93%	0.5158	83.29%
0.1	**0.4765**	**84.63%**	**0.35562**	**89.11%**
0.15	1.2731	72.30%	1.4249	64.19%

Table 5.5: Learning rate parameter

The compression ratio parameter:

Compression_ratio	**FedAvg**		**Proposed Approach**	
	Loss	**Accuracy**	**Loss**	**Accuracy**
5	1.1517	64.44%	0.8718	74.15%
6	1.237	61.06%	0.9598	76.33%
7	1.1942	61.38%	0.8987	78.01%
8	1.1794	65.96%	0.9561	76.84%
9	**1.1206**	**66.26%**	0.8764	76.30%
10	1.4122	49.67%	0.8951	78.20%
11	1.1593	61.93%	0.906	73.32%
12	1.3061	54.21%	**0.9015**	**78.61%**
13	1.4695	38.18%	0.9306	76.92%
14	1.5083	40.83%	0.8891	77.31%
15	1.4053	47.54%	0.887	77.43%

Table 5.6: Compression ratio parameter

The final and optimal parameters:

Parameter	**Value**
number of clients	12
Batch size	36
Epochs	20
Learning rate	0.1
compression rate	9

Table 5.7: optimal parameters

5.4 Results

After testing a number of parameter values and deciding which combination produce the optimal results, in this section we present the final result that each of the two models achieved and we explain the key difference between the two and which model performed better.

First, The table 5.8 shows the results achieved on each epoch by the two methods, then the plot 5.1 shows the deference between the two methods.

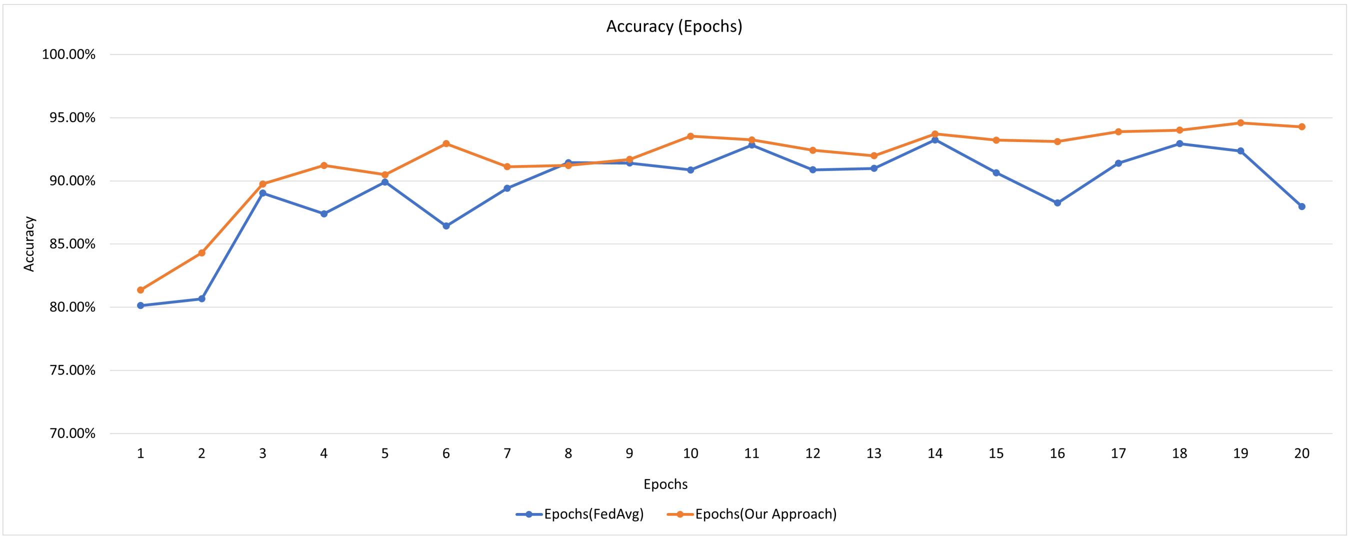


Figure 5.1: The accuracy of our approach vs FedAvg

Our Approach			**FedAvg**	
1	81.37%		**1**	80.14%
2	84.31%		**2**	80.67%
3	89.75%		**3**	89.03%
4	91.22%		**4**	87.40%
5	90.49%		**5**	89.91%
6	92.95%		**6**	86.43%
7	91.12%		**7**	89.42%
8	91.23%		**8**	91.45%
9	91.70%		**9**	91.41%
10	93.54%		**10**	90.87%
11	93.25%		**11**	92.84%
12	92.42%		**12**	90.88%
13	91.99%		**13**	90.99%
14	93.71%		**14**	93.25%
15	93.23%		**15**	90.65%
16	93.11%		**16**	88.25%
17	93.90%		**17**	91.40%
18	94.02%		**18**	92.95%
19	94.60%		**19**	92.37%
20	94.28%		**20**	87.97%

Table 5.8: Our Approach and FedAvg results

5.4.1 FedAvg results

- The initial accuracy at Epoch 1 is 80.14%.
- It shows some improvement over the next few epochs, with fluctuations.
- It reaches a peak accuracy of 93.25% at Epoch 14, and then the accuracy varies but remains relatively high.
- The final accuracy at Epoch 20 is 87.97%, showing some decrease from the peak.

5.4.2 Our Approach results

- The initial accuracy at Epoch 1 is 81.37%, which is slightly higher than FedAvg.
- It demonstrates consistent improvement over the 20 epochs.
- It reaches a peak accuracy of 94.60% at Epoch 19.
- The final accuracy at Epoch 20 is 94.28%, maintaining a high level of accuracy.

Based on the achieved accuracy results, the proposed approach demonstrates better and more consistent performance than FedAvg in terms of accuracy improvement and maintaining a high level of accuracy over the 20 epochs. The stability and consistent improvement in the proposed approach suggest that it is a more effective method in federated learning.

Chapter 6

Conclusion

6.1 Problems and Difficulties

As a research paper, a huge number of articles and other published research papers needed to be fully view and reviewed, one of the biggest difficulties was finding these papers and gaining access to them legitimately, huge number of the papers published in this field are not free, and require some sort of fees or subscription by the publisher to gain access.

Another problem was to fully understand each algorithm we reviewed in this paper, to then re-write the logic behind it, without going in much details about how a certain formula was made. Certain algorithms needed to be put in a more simpler way so that the reader would understand the logic more and have a clear idea of what's the difference between methods.

6.2 Findings

We look for some research and papers related to federated learning that evaluates and compare algorithms, FedAvg is the most used algorithm. here are some findings: FedMa over performs FedAvg but fails to solve complex tasks. FedAvg has better results than FedPer and FedMa from some global and local test-set. FedAvg shows that it achieved the highest accuracy among the federated algorithms. FedPAQ managed to tackle the communication bottleneck problem and reduce the communication load.

6.3 Future work

Next step is to compare between different methods that are examined in this paper, using one or more of the frameworks from chapter 2, by the end of the implementation step we should have our own results based on the comparison that we preformed, then proposing certain methods for different circumstances.

6.4 Conclusion

Federated learning is an immensely growing and vast field that is ever more growing in there passing years, with prominent result, with more developments, advancements and algorithms that are being developed all the time, still it has much more to offer and be discovered. In 2017 google proposed Federated learning which was created to solve certain privacy aspects of machine learning, which it has, but in doing so it created specific problems, such as communication problems and security problems, each algorithms is targeted to solve these problems, some of these issues are resolved by trading off some aspects of the algorithm in exchange for another like increasing accuracy at the cost of communication efficiency depending on what that specific algorithms aims to achieve, FedPAQ for an example is one of those algorithms that aims solves some of the problems federated learning has which such as the communication challenge. As of yet there is no one single algorithm that is able to solve all the problems at once, but who knows what the future holds for this promising rising field of technology.

Bibliography

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics.* PMLR, 2017, pp. 1273–1282.

[2] Oxford English Dictionary. Artificial intelligence definition. [Online]. Available: https://www.oxfordreference.com/display/10.1093/oi/authority.20110803095426960;jsessionid=16DC188B61DB322084D764A7608BCBD0

[3] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[4] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[5] O. Shahid, S. Pouriyeh, R. M. Parizi, Q. Z. Sheng, G. Srivastava, and L. Zhao, "Communication efficiency in federated learning: Achievements and challenges," *arXiv preprint arXiv:2107.10996*, 2021.

[6] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the second workshop on distributed infrastructures for deep learning*, 2018, pp. 1–8.

[7] S. Ek, F. Portet, P. Lalanda, and G. Vega, "Evaluation of federated learning aggregation algorithms: application to human activity recognition," in *Adjunct Proceedings of the*

2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers, 2020, pp. 638–643.

[8] T. Sztyler, H. Stuckenschmidt, and W. Petrich, "Position-aware activity recognition with wearable devices," *Pervasive and mobile computing*, vol. 38, pp. 281–295, 2017.

[9] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1192–1209, 2012.

[10] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7232–7241, 2021.

[11] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, 2017.

[12] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, 2019.

[13] D. Jhunjhunwala, P. SHARMA, A. Nagarkatti, and G. Joshi, "Fedvarp: Tackling the variance due to partial client participation in federated learning," in *The 38th Conference on Uncertainty in Artificial Intelligence*, 2022.

[14] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[15] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proceedings of the National Academy of Sciences*, vol. 118, no. 17, p. e2024789118, 2021.

[16] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *CoRR*, vol. abs/1702.05373, 2017. [Online]. Available: http://arxiv.org/abs/1702.05373

[17] Q. Li, Y. Diao, Q. Chen, and B. He, “Federated learning on non-iid data silos: An experimental study,” 2021.

[18] G. V. Ramesh, G. Chennupati, M. Rao, A. K. Sahu, A. Rastrow, and J. Droppo, “Federated representation learning for automatic speech recognition,” 2023.

[19] V. Kulkarni, M. Gawali, and A. Kharat, “Key technology considerations in developing and deploying machine learning models in clinical radiology practice,” sep 2021. [Online]. Available: https://scite.ai/reports/10.2196/28776

[20] “How big should batch size and number of epochs be when fitting a model?” [Online]. Available: https://stackoverflow.com/questions/35050753/how-big-should-batch-size-and-number-of-epochs-be-when-fitting-a-model

[21] “python - epochs vs rounds in federated learning - stack ...” [Online]. Available: https://stackoverflow.com/questions/71822452/epochs-vs-rounds-in-federated-learning

[22] X. Pan, X. Wang, C. Zhao, J. Wu, H. Wang, S. Wang, and S. Chen, “Usfp: An unbalanced severe typhoon formation prediction framework based on transfer learning,” *Frontiers in Marine Science*, vol. 9, feb 2023. [Online]. Available: https://scite.ai/reports/10.3389/fmars.2022.1046964

[23] “Communication-efficient federated learning via knowledge ... - nature.” [Online]. Available: https://www.nature.com/articles/s41467-022-29763-x