SHANU, SHIPRA. M.S. Improving Federated Learning in Heterogeneous Wireless Networks. (2022)
Directed by Dr. Jing Deng. 53 pp.

The next era of privacy preserving machine learning is built upon the basic principle centered around data privacy. Due to data privacy being the primary concern among data owners, there is an increasing interest in a fast-growing machine learning technique called Federated Learning (FL) [1], in which the model training is distributed over mobile user equipment, exploiting user equipment local computation and training data. Federated learning has emerged rapidly and has been used widely for creating privacy-preserving models by building local models privately. This new sought-out technology does not require uploading one's own private data to a central server to train the models. Rather, computations are moved closer to the data, i.e., a globally shared model is brought to where the data resides. Models are moved towards the device and this allows the models to be collectively trained as a whole. FL has become an answer for working out all the conflicts among data sharing requisite and data privacy concerns, as it sends the models to the data not the other way around. The concept of FL is beneficial in wireless networks where it plays an effective role in training FL models on devices like mobile phones and IoT (Internet Of Things).

To describe FL in simpler words, it is an approach that allows users to train a neural network model without holding client data physically at their locations. It is a type of machine learning where we do not centralize all the data on a single server. The concept of FL has made possible to collaborate data from heterogeneous sources, train the model local at the data location. There are multiple advantages of having the data reside in silos and still various types of analysis can be performed

securely. Despite its many advantages, FL comes with an ample amount of challenges that need to be addressed for effective training and testing such as Statistical and System heterogeneity across all the user equipment's data and physical resources.

FL relies on multiple sources of data and each of the clients is unevenly distributed. They are quite likely generating non-identical independent distributed (non-IID) data. Researchers have worked on multiple algorithms trying to resolve/increase the accuracy of model. However, there are many fields and gaps in the experimental study on understanding their advantages and disadvantages. Most of the previous studies have rather unreal data partitioning strategies among the clients, which are hardly representative of the real-world scenarios [2]. One major issue with FL is the data discrepancy among different locations and users moving among different locations. In this work, we investigate comprehensive data partitioning strategies to cover the typical non-independent identical distribution (i.i.d.) data cases. Moreover, we conduct extensive experiments to evaluate algorithms with various data partitioning strategies and how to recover the loss of accuracy. Our multiple ways of dividing data in non-IID settings brings significant challenges in learning accuracy of FL algorithms. We explore Simple CNN (Convolutions Neural Networks) models for training on data locally to the data zone. CNN models make use of Stochastic gradient descent to minimize the cost function. We alter the different parameters of SGD to learn its effects on the model training.

IMPROVING FEDERATED LEARNING IN HETEROGENEOUS WIRELESS NETWORKS

by

Shipra Shanu

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2022

Approved by

Committee Chair

To my husband and my parents for their continual support of my academic career.

APPROVAL PAGE

This thesis written by Shipra Shanu has been approved by the following committee of the Faculty of The Graduate School at The University Of North Carolina at Greensboro.

Committee Chair _____
Jing Deng

Committee Members _____
Lixin Fu

_____
Minjeong Kim

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to thank my advisor, Jing Deng, for helping me to prepare this Thesis. His experience and advise were invaluable.

PREFACE

Much of the researched I surveyed for federated machine learning was not only new but also quite complex to understand. I was curious to understand it more and more and wanted a way to understand the affects on accuracy for moving and stationary clients and achieve better results for non-IID data settings.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

I.1. Overview

Machine Learning is one of the most sought after technologies in the field of AI (Artificial Intelligence) over last decade. There has been a great amount of research work carried out by the researchers in this field to overcome some of the most important challenges. Machine learning deals with very large data sets collected over time from different sources such as Ease.ml, Machine Learning Bazaar and Rafiki [3]. There has been huge growth in leveraging machine learning services, such as learned index structures and learned cost estimation. Despite huge success and great advantages that the machine learning brings to the table there has also been some constraints that it poses. Machine Learning (ML) effectiveness highly relies on large-volumes of high-quality training data. However, due to the increasing privacy concerns and data regulations such as GDPR (General Data Protection Regulation), training data have been increasingly fragmented, forming distributed databases spread all over the world. Due to these reasons, the raw data are usually not allowed to transfer across organizations/countries. In order to build effective ML models, it is necessary to learn the various features from these data silos. So, it was absolutely necessary for the field of machine learning to overcome these issues of privacy and sensitivity in order to be successful and effective. This gave birth to concept called federated machine learning. Federated machine learning makes sure that the data never leaves the organization or for the matter the components in which it is stored and managed. Federated Learning is part of Machine Learning

techniques, it came into importance due to security and privacy concerns in 2015-2016 [4]. It is a process that enables machine learning models to train from different data sets located in different sites (e.g. local data centers, a central server) without sharing training data. The centralized server will send a local model to each participant before training takes place. After every round of federated training, the participants send back its local gradient to the global model and the server updates it with the average of all the local gradients. One of the biggest challenges in federated learning is distributed data and device heterogeneity. To have a personalized model such as different models created for different participants, statistical heterogeneity needs to be addressed first. This implicitly leads to model heterogeneity. To tackle statistical heterogeneity is to have an individual model for each participant. However, we also need to ensure that the individual model converges to a true global model which is not possible with simple averaging due to client drift.

I.2. Heterogeneous federated learning

Federated Learning involves multiple applications that are from different domains and require to work on heterogeneous client's data-sets. One common feature among these models is that they all share the same centralized global model, where all the parameters are updated by the local models. Recently, there have been some new federated learning frameworks to address heterogeneous clients equipped with very different computation and communication capabilities. We will take advantage of these techniques to train heterogeneous local models with dynamically varying computation and non-IID data complexities while still producing a single accurate global inference model.

I.3. IID vs Non-IID data

FL model may deal with some of the most complex and unstructured data available to train its models. In most cases, we cannot assume and develop inferences after training local models on simplified Identical distributed data. This might hold good in very ideal conditions but model will not be effective in non identical distributed sources. Under this setting, the performances of the training process may vary significantly according to the unbalanced local data samples as well as the particular probability distribution of the training examples.

In Figure I.1 there are 10 data zones labeled from 0 to 9 which are plotted on x=axis. On the y-axis, we have plotted total count of labels assigned to a particular data zone. Each of these zones contains 80% of one label and the rest 20% data is equally contributed by all other labels. It can be seen that zone 0 has 80% of label 0 and rest 20% equally distributed. All the other data zones are also distributed similarly.

In Figure I.2, we can see that the labels are uniformly distributed among the data zones.For the IID data partitioning, all the labels are equally distributed for each data zone.

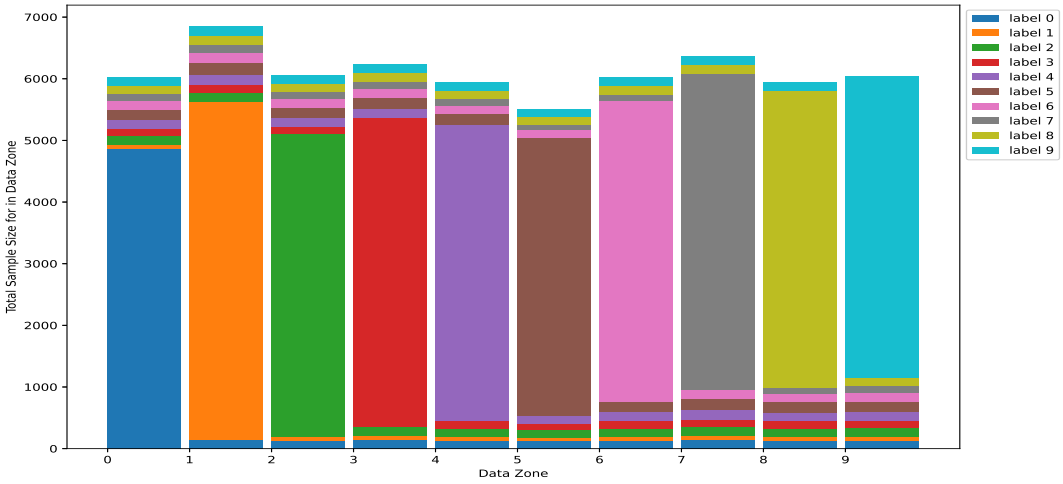

Figure I.1. Distribution of labels among data zones on basis of non-IID settings

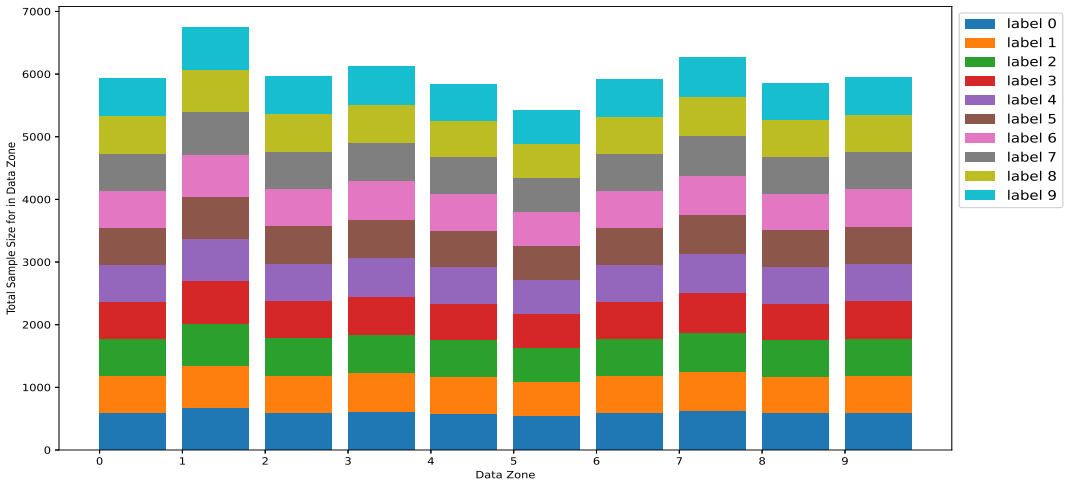

Figure I.2. Distribution of labels among data zones on basis of IID settings

CHAPTER II

RELATED WORK

In recent times in the field of Machine Learning, there is emergence of concept called Federated Learning which has been topic of extensive research among the developers community since 2015 and 2016. It initially set its foot in the field of Telecommunication with its first publications on federated averaging in telecommunication settings [5]. One of the most profound areas of current research is reduction communication burden which arises from the FL concept. In 2017 and 2018, there have been many publications that emphasized on the development of resource allocation strategies. Most of the publications are specially based on the gossip algorithms to reduce the communication requirements of different nodes involved in FL process. There are also publications that covered the topics like characterization of the robustness of the FL model to differential privacy attacks . Federated Learning Process requires very high bandwidth to transfer parameters between centralized global model and client models. Reduction of bandwidth during training through sparsification and quantization methods for FL was one other topic taken up by other researchers. In this scenario various FL models are sparsified and/or compressed before they are shared with other nodes. In order to develop a very effective Deep Neural Network architecture for the ultra-light FL model training it is necessary to be very energy efficient as well as cost efficient. Otherwise the model will not sustain. It also necessary to make sure that number of transaction between the Client model and global model is as low as possible. There are many other issues related to privacy that form as a concern during the training

process.

On the similar lines of Federated Learning and with some improvements to already existing problems with FL, there is another learning framework named Assisted learning was recently developed to improve each client model learning capabilities without transmitting private data. It also improves the capabilities to transfer model parameters and learning objectives. Federated learning often requires a central controller or global model to orchestrate the learning and optimization FL process. compared to Federated model, Assisted learning goal is to provide parameters and protocols to to other model client who are part of the learning process to optimize and learn among themselves without a global model.

II.0.1. Use cases

Federated learning is mainly used when data from an individual or big corporation needs to used for training to build a predictive model but due to sensitivity of the data it cannot be shared over network or with other people (e.g., for legal, strategic or economic reasons). However it is necessary that these people or organizations are well connected to internet for the models to move to their location for training.

II.0.2. Transportation: self-driving cars

Self-driving cars are heavily dependent on the some of the predictive algorithms which internally use machine learning techniques. Some of the core functionality of the self driving car are, computer vision for analyzing obstacles, machine learning for adapting their pace to the environment (e.g., bumpiness of the road). In the future of the self driving car world there we will need the cars to respond to the real situations very quickly. Current cloud approach to receive instruction will not be enough to support the error free functioning of self driving

car. FL can represent a solution for limiting volume of data transfer and accelerating learning processes. [6]

II.0.3. Industry 4.0: smart manufacturing

In Many Industries, there is need for a algorithm based research for the demand and supply. Many of the industries use the machine learning techniques to predict the use, requirement and current, future sales of the items. To improve the efficiency and effectiveness of the production units of many industries there is need for adoption of Federated model training where data is sensitive. In addition, FL also implemented for prediction to support Smart city sensing applications. Industries help grow the cities. Considering this in mind will help grow the city and civilization in general.

II.0.4. Medicine: digital health

One of the most effective areas where the federated machine learning techniques are very commonly used is the Medical science. There are number of medical science experiments and discoveries happening around the world. Los of the medical companies perform medicine trials on the animals before trying on the humans. All of these results and calculations are limited to be used by the same organizations due to privacy and sensitivity concerns. There are huge number of patients for terminal diseases like cancer are being treated. the data related to the patients are not shareable to the other organizations due to patient privacy. This field of Federated Learning has made possible to access all such data from all over the organizations base on Federated Learning technique. This will help in development of newer medicines and also maintain demand supply chain. Nature Digital Medicine published the paper "The Future of Digital Health with Federated Learning". [7]in September 2020, in which the authors explore how federated

learning may provide a solution for the future of digital health and highlight the challenges and considerations that need to be addressed. Recently, around representatives of 20 different institutions around the world examined the utility that trained AI models using Federated Machine Learning. In a paper published in nature medicine Federated learning for predicting clinical outcomes in patients with COVID-19 [8] the organizations showcased the requirement of oxygen by the patients of COVID-19. The federated AI model was able to precisely predict it. Furthermore, in a published paper A Systematic Review of Federated Learning in the Healthcare Area: From the Perspective of Data Properties and Applications [8], the authors trying to provide a set of challenges on FL challenges on medical data perspective.

II.0.5. Information Technology

Looking at the advantages of the federated Learning techniques, Google introduced FL in 2017 on the the mobiles devices to train a machine learning model about the data usage while keeping the raw training data on each user's device, decoupling the ability to do ML from the need to store the data. This has allowed Google to access data from millions of mobile devices to perform many such training. This would help in better tar-getting the customers with products. [9]

II.0.6. Previous Research vs. Our Research

Many previous researchers have worked in the field of Federated Learning and lot of work has been committed by the research community. MNIST data-set if one of the very famous and most used data sets to perform simple federated Learning Experiments. Previous, researchers have trained their models different types of data IID, non-IID. In our experiment we have designed algorithms to partition the MNIST data set with different variations of non-IID fashion. This led us to explore

the behaviour of simple CNN models with the different data sets. In Decentralized federated learning we have seen a model train at on node and provide the inputs to central node for its training. In our experiment we have the local model(moving clients) are not stagnate at one data zone rather they move to different data zone after training and testing on one data zone. In our experiment we are trying to study the behaviour of the client when they move from one data zone to anther Non-IID data zone. Most of the existing models use gradient descent for minimizing the loss function. Gradient Descent plays a very important role in identifying the next iteration. Good Gradient helps in convergence of the graph more quickly.

CHAPTER III

FEDERATED LEARNING

III.1. Overview

Federated learning (FL) is a form of a machine learning applications where multiple clients like mobile devices or whole of the organizations collaboratively train a model over a decentralized database [10]. FL has the concept of centralized Global Model and Local Client Models that trains on the decentralized data from different organizations or devices.

Consider there are N organizations or N people generating data on mobile devices. FL model wishes to train on this sensitive data. In a conventional machine learning model all the data will be consolidated at one place and an ML model trains on it. This leads to an issue with data privacy. An FL System maintains the privacy of the data and data never leaves organization but rather a local ML model is trained on the data independently and then learned features/parameters are shared to the global model.

In this work, we consider MNIST data set that is randomly divided among multiple data zones to represent N wireless devices data. In this the local model is initialized with the initial parameters though initialization process. Let us consider initial parameters being represented by ω_t in t^{th} rounds of training from Global model, when t=0. After receiving the initial parameters local models train and update with n data let us consider intermittent local model to be γ_b^{t+1} for b user data-zone,

$$M_s^{t+1} = \frac{1}{K} \sum_{k=1}^{K} \gamma_k^{t+1} \tag{III.1}$$

where M_s^{t+1} denotes the model parameters obtained in $(t+1)^{th}$ round

After the model training the parameters aggregation with the global model plays very important role. We copy the parameters form the local trained models to the global model.

$$argmin_{g_{t+1}} \sum_{k=1}^{K} (\alpha_k^L(g_t, \gamma_{t+1}^k)^2 + \mu(g_t, M_s)) \tag{III.2}$$

where g_t, γ_{t+1}^k denotes global parameter and local parameters of user k at the $(t+1)^{th}$ iteration. L Function is used to calculate the difference between the two models before updating. α_k is the parameters of the global model for user data-zone k. μ is used to proportionate the share of local model in the global model.

We find the difference between the global model and the local model.

$$[S_K^i]_{i=1}^1 = ||g^i - \gamma_k^i|| \tag{III.3}$$

where g^i denotes the parameter at the of the global model at the t^{th} iteration. Similarly, γ_k^i is the parameter value of local model of user k^{th} data-zone at t^{th} iteration.

After obtaining the difference between the local model and the global model for user k^{th} data-zone. We repeat the above mentioned procedure to obtain the parameters of each user data-zone.

We derive the corresponding gradient for all the users, as follows :

$$\Delta = \sum_{k=1}^{K} \alpha_k (g_t - \gamma_{t+1}^k) + \mu(g_t - M_s) \tag{III.4}$$

For all the K users who participated in the training, the algorithm optimization process is shown in the following equation, where λ denotes the step size. The global model g_{t+1} is finally obtained after the t+1 iteration.

$$g_{t+1} = g_t - \delta(\sum_{k=1}^{K} \alpha_k (g_t - \gamma_{t+1}^k) + \mu(g_t - M_s)) \tag{III.5}$$

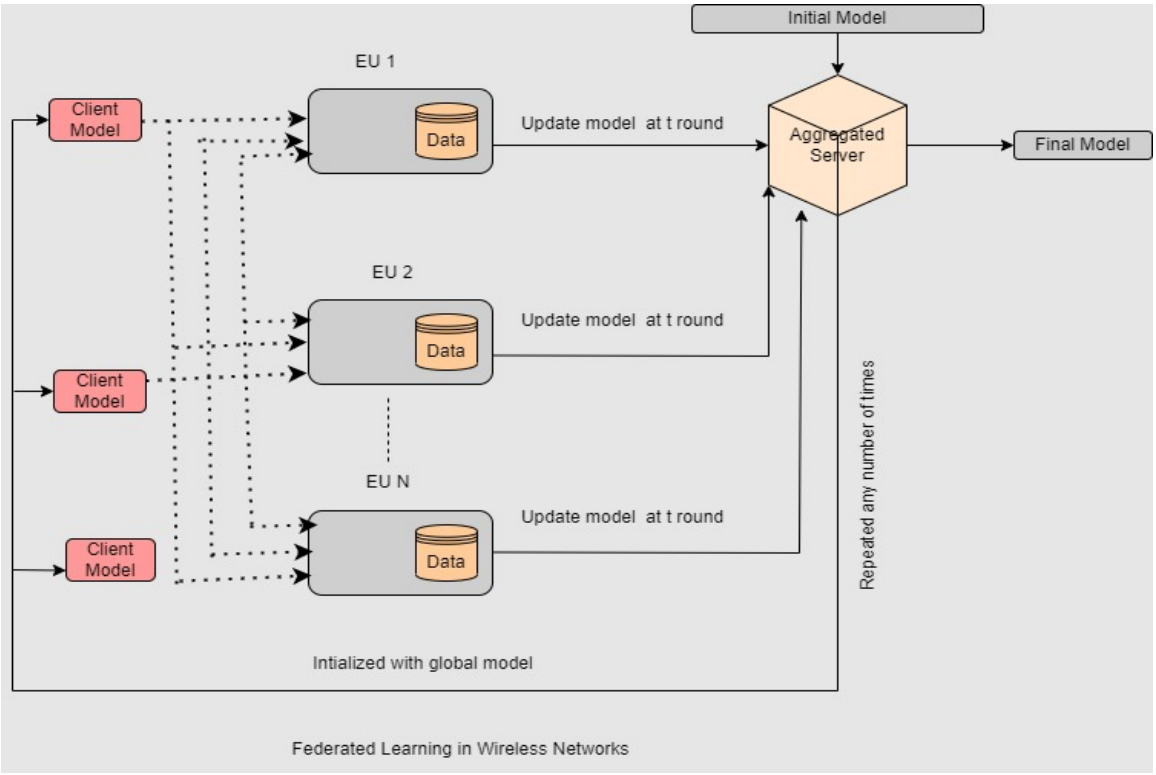

Figure III.1. Flow diagram for simple federated learning process

III.2. The Life cycle of a Model in Federated Learning

In Federated Machine Learning it is important to identify a proper problem that fits into its use case and data that supports the use case. We also need to identify an algorithm that would train the model effectively. We need to correctly identify the features that needs to be considered for training. There are many machine learning methods like clustering, classification, deep learning. We need to select these methods depending upon the problem in which category they fall.

There are several important stages of a FL model training. Some of them are listed below.

1. **Problem identification:** It is one of the major steps in Federated Learning Process. There are many unresolved use cases when it comes to model being trained to predict, detect or identify things correctly.

2. **Clients:** Selection of client that will provide data is necessary. In FL store the data at the client side and training mostly takes places at the client location so it is absolutely necessary that the client side have some computational capability. Let us consider the use case of mobile phones, there are many apps running on the mobile. In most of the cases the apps also store the data that can be used for training e.g, text message, photos etc. However, in some cases additional data or metadata might need to be maintained, e.g. user interaction data to provide labels for a supervised learning task.

3. **Simulation prototyping:** This can be an important step if we are experimenting on a sensitive data. It is always good to have a prototype model architecture testing done before working on actual data. However this is step is not always necessary. It can be removed if not needed.

4. **Model training in Federated Architecture:** Federated Model

Training initialises multiple tasks like Global model initialization, copying all the parameters to all the local model clients from global models etc. Local models are initialized with different hyper parameters based on the types of training and optimizations, each of the model client needs to follow.

5. **Model Evaluation:** After the Local models training for required number of round, the models are evaluated based on the accuracy at which they are able to train. The Analysis on the training can include metrics computed on standard data sets or derived data sets. In this scenario, it is extremely important to populate the training data properly. In the evaluations process all the good data is selected and bad data is removed from the model training. Train/Test Accuracy portrays the efficiency of the model.

6. **Deployment:** This is the final step of the Federated Learning. After the model is selected during the prototype run and we are satisfied with the results, this model is selected for the deployment onto to the target client locations to train on actual live data. The deployment of the models are done in step wise manner. We can also run a quality test before deployment.The specific launch process for a model is set by the owner of the application and is usually independent of how the model is trained. In other words, this step would apply equally to a model trained with federated learning or with a traditional data-center approach.

T̂he end goal of all the above steps is to see a fully working federated models deployed on the client's devices that would train on client data and revert with the learned parameters to global model for aggregation. [11]

III.3. A Typical Federated Training Process

Federated Learning is a process of moving model to stagnate data approach. Its way to build a prediction model with sensitive data. FL has lot of advantages

over other Machine Learning techniques in maintaining privacy, data protection.

In a typical Federated Machine Learning, we have a server (Global Model) , a number of client nodes (Client Models) that trains on the Data zones (Enterprise Data sets). A client node trains on the data zone and passes on the training parameter to the central server(Global model) for it to learn. After each round of training the Global model is updated with its parameter. This process is repeated with all the client models and finally a trained Global model is ready for testing.

1. **Client:** The server samples from a set of clients meeting eligibility requirements. For example, mobile phones might only check in to the server if they are plugged in, on an unlimited WiFi connection, and idle, in order to avoid impacting the user of the device.

2. **Broadcast:** The selected clients download the current model weights and a training program from the server.

3. **Client computation:** Each selected device locally computes an update to the model by executing the training program, which might for example run SGD on the local data (as in Federated Averaging).

4. **Aggregation:** The server collects an aggregate of the device updates. For efficiency, stragglers might be dropped at this point once a sufficient number of devices have reported results. This stage is also the integration point for many other techniques which will be discussed later, possibly including secure aggregation for added privacy, lousy compression of aggregates for communication efficiency, and noise addition and update clipping for differential privacy.

5. **Model update:** The server locally updates the shared model based on the aggregated update computed from the clients that participated in the current round.

III.4. Experiment Setup

III.4.1. Experiment Data(MNIST)

Experiment is primarily focused on studying the behaviour of the federated Learning models with non-IID data when the local model are mobile among the data zones. For this experiment, we have considered MNIST data set. MNIST data is group of hand written images with a range of 0-9. Altogether, it has a total of 60,000 handwritten images of digits in the training data sets and a total of 10,000 images of test data sets. MNIST is a subset extract of the another large data set NIST. All the digits in the MNIST are normalized to a certain size and fixed-sized to center of the image. the original MNIST data set are black and white images of digits and the digits are normalized to fit in 28*28 pixels while maintaining the aspect ratio. Each image generate 28*28 array that be flattened into 784 dimensional vector. [12]

III.4.2. Data Partition

A number of algorithms are defined to divide and assign the data set in number of so called data zones each corresponding to one user/one organization in wireless network. In this experiments, Dz data zones are considered, can be viewed as the clients that provide data for training. There are different fashion in which the MNIST train and test data sets can be divided for the experiment. some of them that we have used for training purposes are listed below.

Training Data set partition

1. IID data set Partition: We will divide the entire MNIST data set into multiple IID data sets. The data set (60,000 training sample) is equally divided among the data zones. All the data zones get equal number of 0's and 1's and so on.

2. Non-IID data set: In this case, we divide the entire MNIST data set into

either completely biased with one label for one data zone or the data zones have unequal number of labels shared among the data zones e.g. unequal number of 0's or 1's and so on.

3. Percentage-non-IID data-set: In this case, We divide the entire MNIST data set into data zones that have some predefined percentage of data bias. e.g., for a 70 percent non-IID data will have 70 percent of one label of data and rest 30 percent of data are equally divided from all other labels.

4. Random-percentage-non-IID data set: In this case, We divide the entire MNIST data set into data zones are divide with uneven percentage of biased data. The algorithm randomly picks up a percentage and that percentage of one label of data is assigned to a zone, the rest of the labels are equally divided. Same thing is repeated for other data zones.

III.4.3. Experiment Algorithm

In this experiment we tweaked existing Federated learning algorithm to create something called HeteroFL Algorithm for training/testing. This algorithm spin up multiple local clients model(C) for the training purpose and a global model for training and testing. Each of these client models initially selects a data training zone e.g. a mobile device or Organizational data. the client model trains on the data for certain rounds. At the end of each round the client models update the parameters to global model. We modified this algorithm to train a client model on a particular data zone for some rounds and then move to another data zone for training. Our goal is to improvise the federated training process in such a way that the moving client model also have good average accuracy.

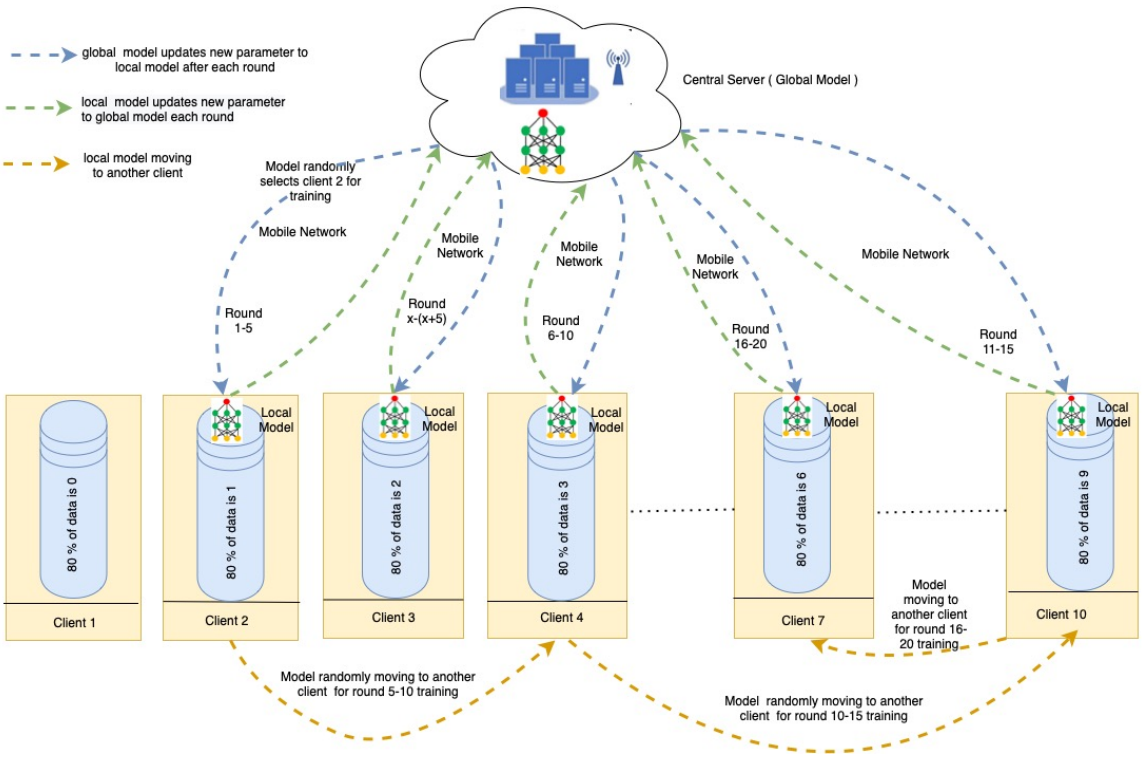

Figure III.2. Federated Learning experiment flow for a moving model client

Algorithm

HeteroFL: C is the total number of client Models, B is the batch size, R is the total Rounds, E is local training Epochs, lr(n) is the learning rate. Dz is the data zone.

Simple CNN

We have utilized Simple Convolutions Neural Network Machine Learning Model for our experiment which is a very widely and deeply used for image recognition and classification techniques today. Yann LeCun, director of Facebooks AI Research Group, is the pioneer of convolutional neural networks. Initially, he had built one of the first convolutional neural network called LeNet in 1988 [13]. It was

Algorithm 1 Federated Machine Learning with moving client

Initialize $G_t \leftarrow \theta_t$
$C_m \leftarrow C_0 ... C_t$ ▷ Randomly select a Client
for <Round 1 to r > **do**
    $D_z \leftarrow D_0 ... D_t$ ▷ Randomly select Data zone
    $G_r < -G_r - l_r(n) L_r(G_t, b)$
    Client-Train(G_t, G_r, b, E)
end for
procedure CLIENT-TRAIN(G_t, G_r, b, E) ▷ Calculate Gradient
    for <local Epoch 0 to E> **do**
        for <batch size > **do**
            $G_r < -G_r - l_r(n) L_r(G_t, b)$
        end for
    end for
    return G_r ▷ Returning Global model with new parameters
end procedure

used to recognize characters like reading zip codes and digits. As the CNN model require very large amount of data to be able to train significantly and more computing resources for the classification and identification, It remained in only limited use for the postal sectors.

Normally Neural networks deal with lot of matrix manipulations like multiplications, but in case of CNN models it uses a special technique called Convolutions. In the field of mathematics Convolutions are mathematical operations with two functions together produce another function that is used to express how the shape of first one is being effected by the other one.

A CNN model is like a feed forward Neural Network that expresses the image data into grid topology. These grids are knows as ConvNets. The ConvNets are further used to analyse and classify the data depending upon the grid features. In other words In CNN, all the images are converted in the form of an array of pixel values.

Altogether the ConvNets play roles in reducing any image into a form that is easier to process without having to let go any features that play important role in identifying the image. There are so many hidden layers in the Convolution Neural Network the simplifies the information or feature extraction from the image. Some of the important ones are Convolution Layer, Relu Layer, Pooling Layer and Fully connected layer.

Relu comes into picture after the feature maps are extracted. It stands for Rectified Linear Unit. The feature map is passed as an input to Relu layer. It considers each element and sets all the negative pixels to 0. The out of Relu layers is rectified feature map.

After the rectified feature map is generated, it is sent through pooling layer to extract pooled feature map. It is then flattened to convert all 2 Dimensional arrays into single linear vector. Finally the Fully connected layer identifies the image. We can visualize this process through an image [14]

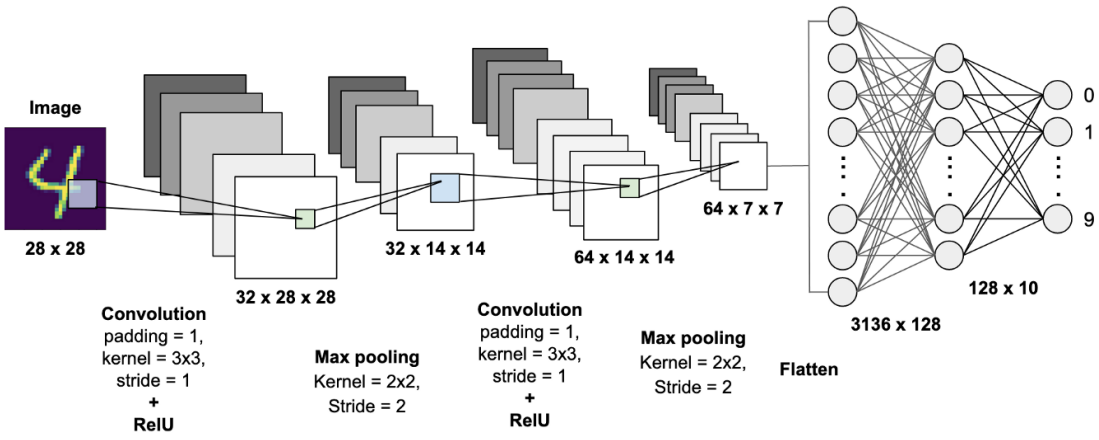

Figure III.3. Processing a digit 4 using Convolutions Neural Network

III.4.4. Client Model Types

Experiment considers moving or stationary client models that either move between the different data zones or stay in the same data zones training for all the rounds of training.

Stationary Client

There are multiple User Data zones but one stationary client model Trains/Test in same data zones only for all rounds. 50 round training, Client Model would train on Data zone 3 for all 50 rounds

Moving Client

MNIST data sets are divided into multiple non-IId data zones. The moving clients randomly choose a data zone for training for certain number of rounds. After that the client model chooses to move to another data zone for training. For 50 rounds of training: Client Model will train in Data zone 3 for 10 rounds and move to Data zone 7 for next 10 rounds and so on to complete 50 rounds of training.

The Global model Update remains same in the both the process. Global model is always updated after every rounds of training with same data zone or different data zone.

III.4.5. Accuracy Calculation

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{Numberofcorrectprediction}{totalnumberofpredictions}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

For an instance, if the number of test samples is 1,000 and global model identifies 952 of those correctly, then the model's accuracy is 95.2 percent.

III.4.6. Global Accuracy

Global Accuracy is calculated using the updated global model. After all round of Client model training is done the global model is updated with all the training parameters from client model Accuracy is calculated with the updated global model. This gives the final accuracy and correctness of the Federated Model training. The accuracy calculation mechanism is same as the one that is calculated at the local data zone level.

III.4.7. Experiments

We have designed the experiments in a such a way to first plot the straight forward graphs for the clients models training in IID, non-IID, variable non-IID data zones while the clients it self be either moving or stationary among the data zones.

Experiment 1 : Effects of Different types of Data on Accuracy

Accuracy graph differences for different data partitions when they are trained for 250 rounds for IID and non-IID data. Our primary goal is to check amount of accuracy drop in moving client model when the data is IID and non-IID. As per our expectation the graph related to moving client should not be very smooth and we should be seeing oscillations in the graph as it changes to new data zones after every 10 rounds. In this part we will also identify number of rounds these graphs take to reach 90 percent accuracy.

Figure III.4 The blue line shows the Accuracy graph for a stationary client model for 250 rounds for different data zones. Type of data is in this example is IID.

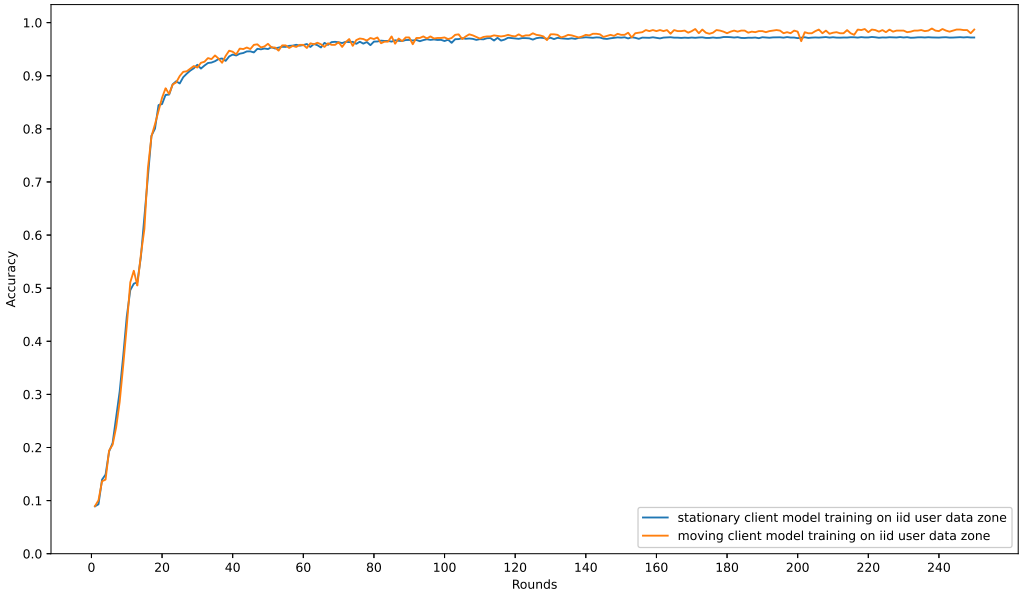

Figure III.4. Blue: stationary client for train/test on IID data for 250 rounds, Orange: moving client for train/test on IID data for 250 rounds

The is the most ideal situation when User Data zone is uniformly distributed with all the labels in equal quantity. The graph is very smooth and we don't see any much of deviation from the trajectory or drops in the accuracy in between the training process. The orange line is plotted for a moving client model. Model Client behaviour changes and now it trains for only 50 rounds in one IID data zone, then it moves to next IID data zone. Global model gets the parameters added in each round and accuracy calculated. We do not find much of difference between these two graphs even with moving/stationary clients as the data sets are identical in every data zone.

In Figure III.5, We considered the above experiment with the 80 percent

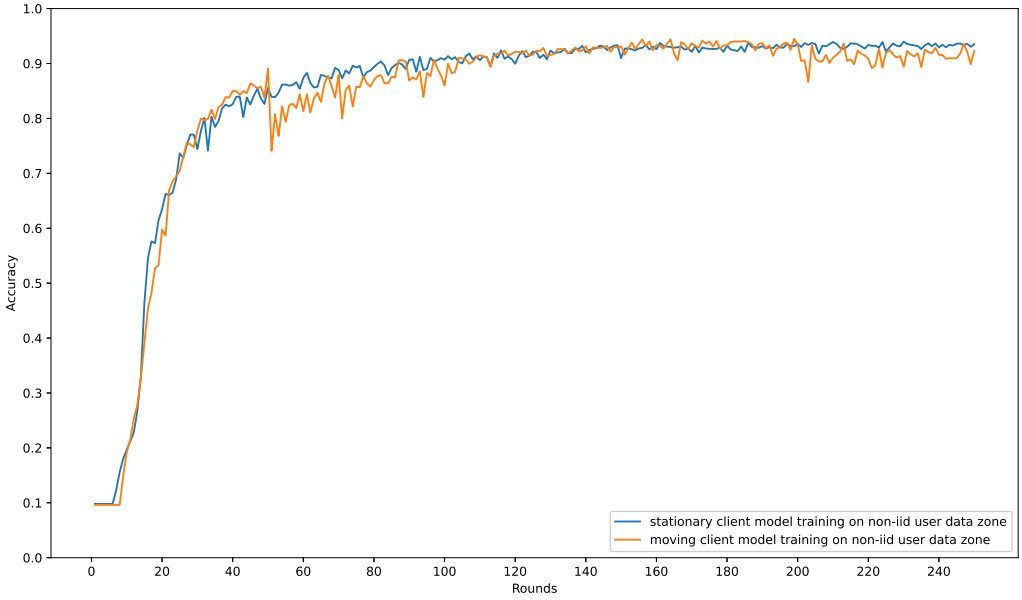

Figure III.5. Blue: stationary client for train/test on non-IID data for 250 rounds, Orange: moving client for train/test on non-IID data for 250 rounds

non-IID data. Since MNIST is a shared data set that is considered to be divided amount different zones. We have created algorithms to partition the data with required percent of non-IId (e.g. 80 percent one label and remaining 20 percent IID data). Model client for blue line is trained on non IID data for 250 rounds. Initially accuracy is low but eventually the model reaches to accuracy level more than 90 percent after 100 rounds of data. In the orange graph the model client is moving so it trains 50 rounds in zone and then moves to another zone for training. If we observe whenever the model client changes the data zone there is huge drop in the accuracy ever after number of trained rounds. We also observe that the graph is not very smooth and have lot of curves.

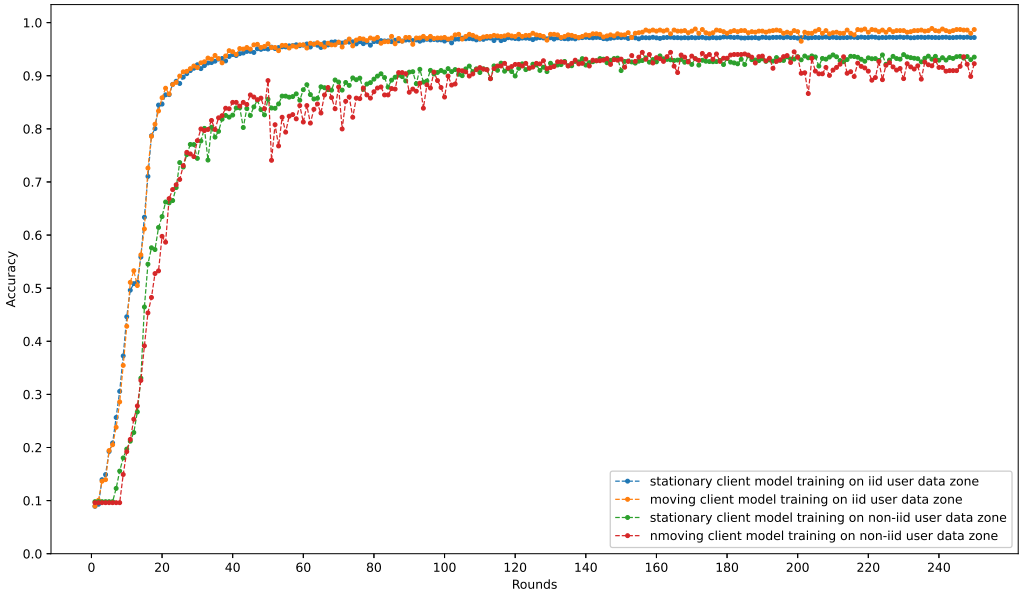

Figure III.6. Accuracy graph with epoch=1 and batch-size=128. Client training with different variations (IID, non-IID) data.

In order to draw parallel among the above graphs with each other, we will merge the graph into one (see Figure III.6). Firstly, we observe that the accuracy graph of the non-IID data is always lower than the IID data. IID data graph do not provide much of difference in graphs. Non-IId graphs are very different from each other in terms of smoothness. We know if the graph is not smooth is not good sign for Machine Learning model.

In TABLE III.1, It shows the accuracy under different Epoch and batch Size. In an IID data zone, the FL model trains with very accuracy rate. The Accuracy rate remains unaffected even if the model clients are moving from one data zone to another. However, the same trend is not seen when the data zones have a non-IID

Table III.1. Analysis of the Accuracy under different Epoch and batch Size

Setting	Epoch and Batch size	Average Accuracy
IID	epoch=50 and batch size=64	93.8
non-IID(80 percent non-IID)	epoch=1 and batch size=64	61.5
non-IID(80 percent non-IID)	epoch=50 and batch size=64	92.3
non-IID(80 percent non-IID)	epoch=1 and batch size=128	59.4
non-IID(80 percent non-IID)	epoch=50 and batch size=128	91.0

data set or variable percentage non-IID data set. In this case the, the table shows that when the data is non-IID and clients are moving the average accuracy falls almost by 30 percent from that id IID data setting. We also notice that when the epoch is increased to 50 from 1, the FL model trains with much higher accuracy. When the batch size increases, we observe a decrease in the accuracy level. We can infer following things from the above table,

1. Accuracy decreases when the training data is non-IID.
2. Accuracy increases when we increase the epoch.
3. Accuracy decreases when we increase the batch size.

Experiment 2 : Effect of Epoch and Batch size on Accuracy

There are multiple factors that control the model training process. Some of the more visible and important ones are epoch, batch size of the data, learning rate of the model. In order to study the effects of these factors we have trained the clients models with these variations we have tried to generate graphs and present our case study for different epochs and batch sizes. Our goal is to identify if the changing epoch and batch size of the training effect the accuracy graph. The Model client is moves around to different data zones during all the experiments in this section. Also, to find out the drop in the accuracy when there is a moving client and the effects on the accuracy if we update the batch size and epochs.

In this experiment we set up a moving client for a total of 100 rounds of

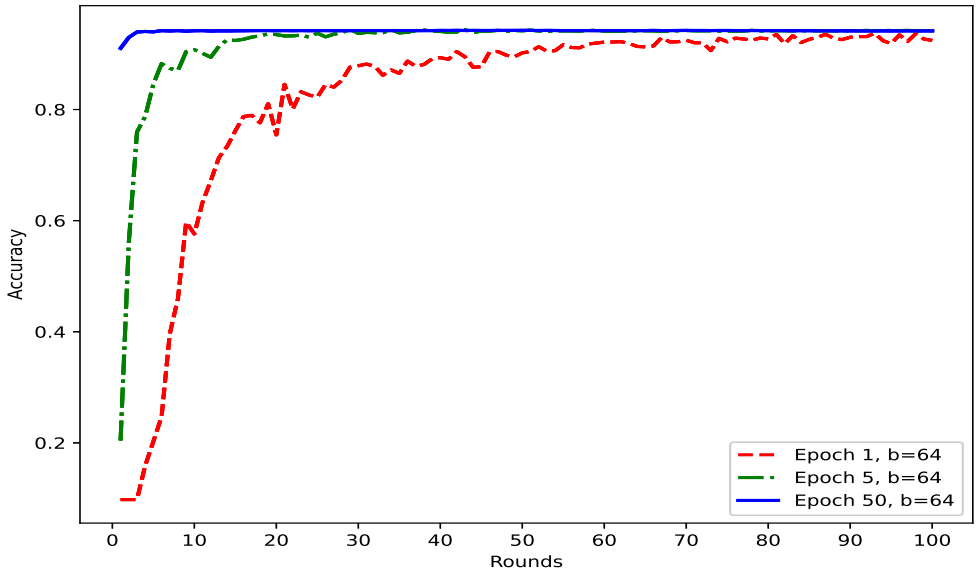

Figure III.7. 100 rounds of client training on non-IID data for different epochs and batch size 64

training with two different batch sizes and multiple epochs to understand the behaviour of the model when the batch sizes are varied. We observed that the lower batch size lowers down the effective accuracy for a given epoch (see Figure III.7). Similarly the increase in the epoch increases the accuracy of the model in training and testing. To understand this behavior little more we can consider the average accuracy vs the epoch graph. On the y-axis the graph represents average accuracy generated from 50 rounds of client model training and x axis represents the rounds of training for which the graph is generated. Different lines in the graph represent accuracy for a given epoch value. The same experiment is conducted for batch size of 128. Moving client differ in their accuracy graph for different epochs and batch

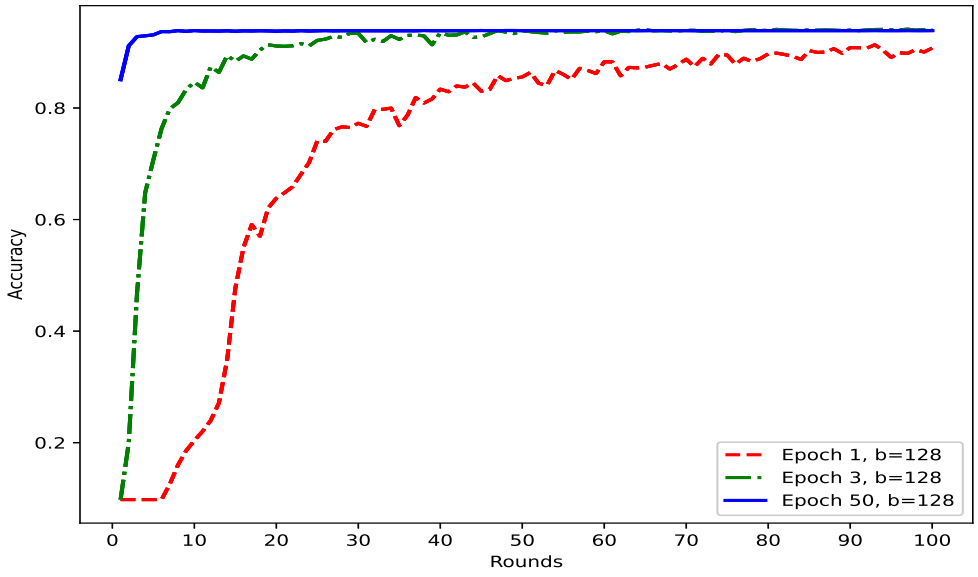

Figure III.8. 100 rounds of client training on non-IID data for different epochs and batch-size 128

sizes. As expected Accuracy graph with epoch=1 has the lowest average accuracy as the initial accuracy is very low. Since the epoch directly influence the number of training cycles the accuracy is very high even for initial training cycles (see Figure III.8).

Batch Size also play a big role in the model training and accuracy calculation. As per observation and experiments we see that higher the batch size the accuracy rate drops lower for the same epoch and rounds of training. We experiment batch sizes of 64 and 128 for 50, 100 and 150 rounds of training. We observed that the number of rounds it takes to reach 90 percent of average accuracy for the different batch size in second rounds and fourth round respectively.

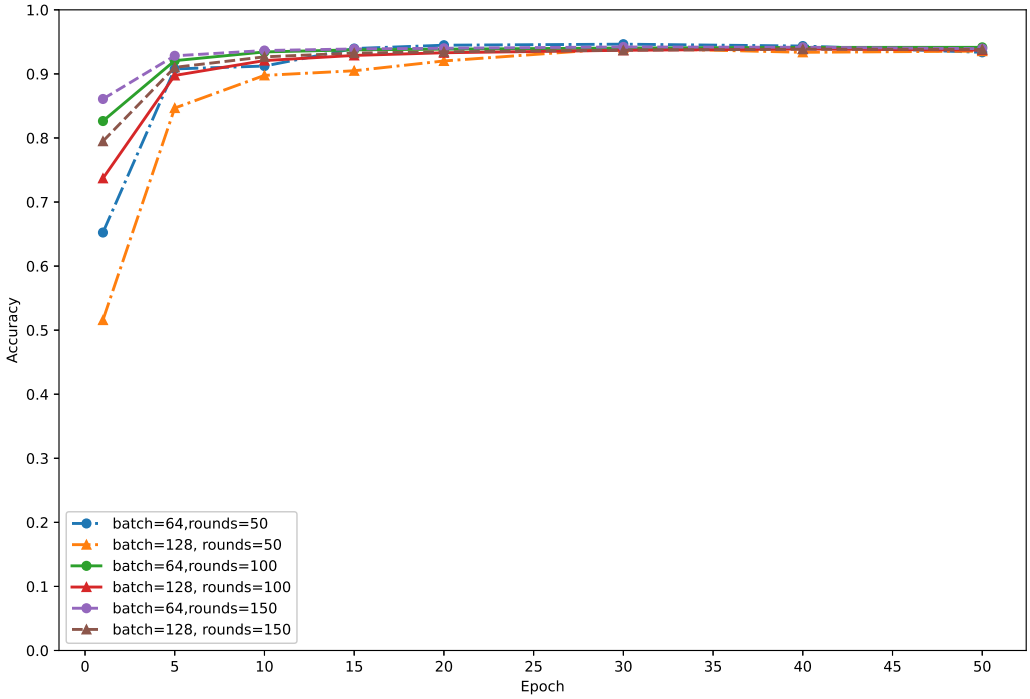

Figure III.9. epoch vs Accuracy plot for 50,100,150 rounds of training

In Figure III.9, Average accuracy when the epoch is 1 is very low even with the increase in the batch size. In this case the accuracy level is much higher. For the same data if we increase the batch size the average accuracy drop for epoch 1 and similarly for the other epochs. we have tried to bring out the relationship between epochs and accuracy when a client is training in a non-IID data-set setup. In this case also we are presenting the Model client to be either moving and non moving. We observe that the moving clients performs differently with increase in epochs than the stationary clients. we also observe the number of rounds it takes for the clients to reach 90 percent accuracy.

Experiment 3: Training Loss Function

Loss Function play a very important role in evaluating the efficiency of the Federated Machine learning. It is a method of evaluating how well a machine learning algorithm models the featured data set. In other words, loss functions are a measurement of how good the model is in terms of predicting the expected outcome. Training loss is calculated and validation and its interpretation is how well the model is doing for these two sets. Accuracy is calculated in percentage but loss is the cumulative summation of the errors that is made during the training and validation phase. Ideally, we should be expecting that the training loss reduces after each rounds of training.

The main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as back propagation in neural networks.

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of miss-classification is calculated. Here we would be only considering the training loss in each round.

In Figure III.10 we are plotting training loss of model client in each rounds of training for a given epoch and batch size. For IID data, the loss graph is very smooth and keeps decreasing. In this experiment we will be dealing with non-IID data. The model client is also going to be moving across the data zones to make the client models and data zones unpredictable. We see that the accuracy drop is

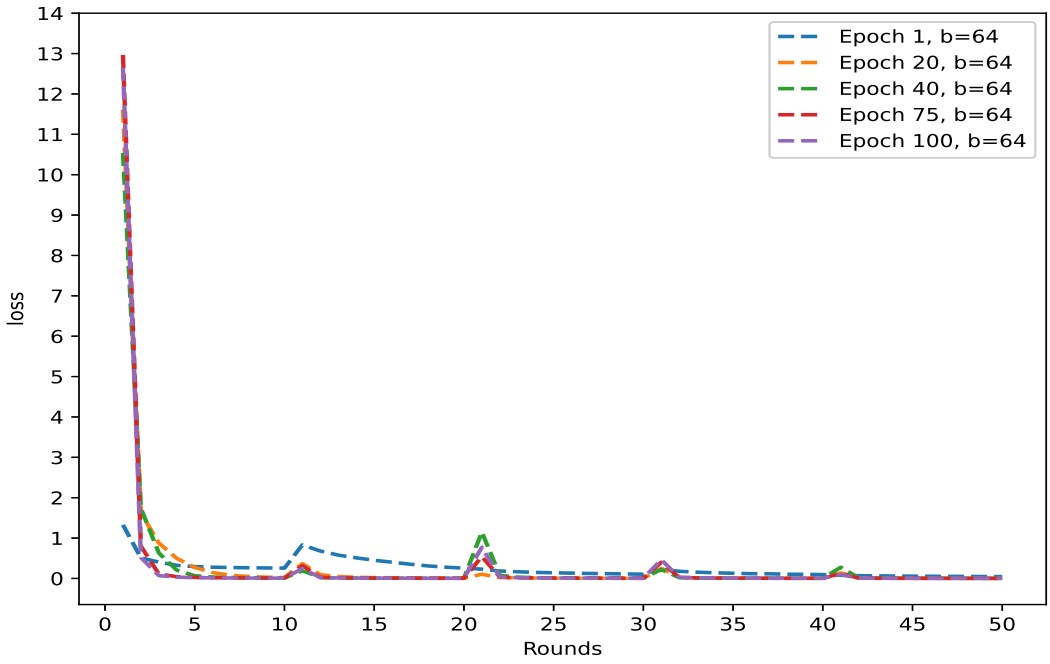

Figure III.10. Loss function of a moving client for 50 rounds for variable epochs and batch size 64

relatively very high for moving clients so one way is to plot the loss function graph for the client. We calculate the loss for the total number of epochs to identify the total loss in one round. Now we plot that against rounds for a moving client with variable epoch and batch size. AS per our observations the epoch loss is very minimal if the epochs are higher. In initial rounds we observe that the training loss is high and eventually it lowers down with the number of rounds progresses. We have tried to study the loss function against the batch size and definitely there is reduction. As we can observe from the graph that there are sudden spikes after rounds 10 or round 20 and so on. This happens when a moving client trains in one

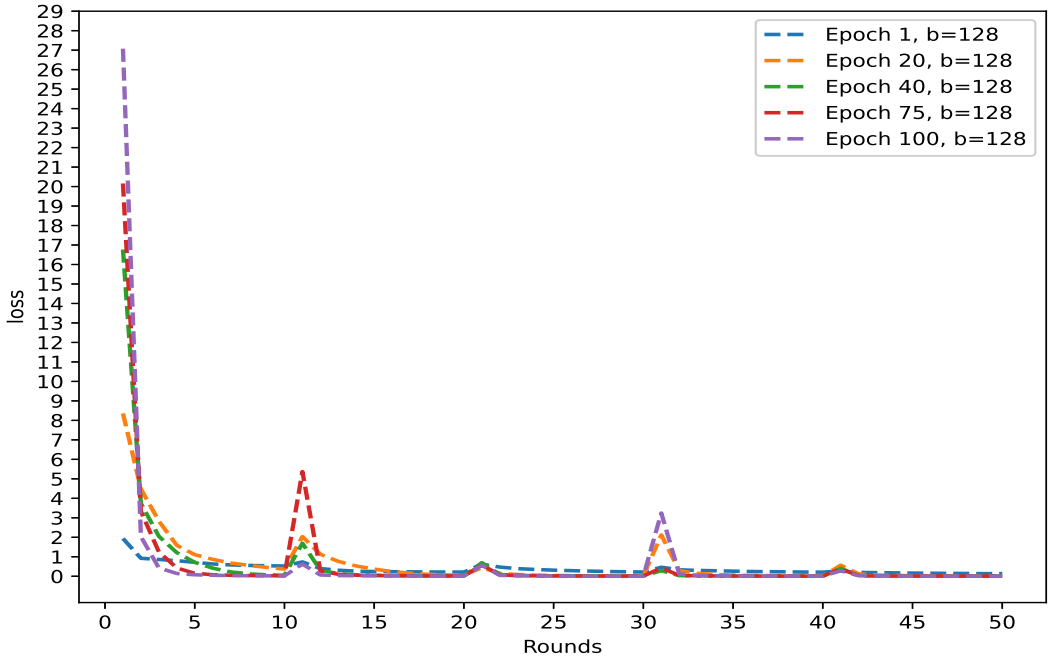

Figure III.11. Loss function of a moving client for 50 rounds for variable epochs and batch size 128

zone for 10 rounds and travels in another another zone we see a spike. We have similar plot for loss function when the batch size is 128 (see Figure III.11)

In TABLE III.2, Evaluating the difference in the accuracy when the non-IIDness of the data is varied.

Experiment 4: Variable non-IIDness

In Figure III.12, we are plotting the average accuracy of 5 trial runs for a moving client with different epochs and batch size. We very clearly observe that the average accuracy if very less for epoch 1 and batch size 128. In most of the trial runs a moving client with higher batch size and lower epoch performs very badly

Table III.2. Evaluating the average loos of the accuracy during the model client training

80 percent non-IIDness	**average Loss for 50 rounds**
IID (epoch=10 and batch size=64)	0.1
non-IID(epoch=1 and batch size=64)	0.2
non-IID(epoch=5 and batch size=64)	0.2
non-IID(epoch=10 and batch size=64)	0.2
non-IID(epoch=1 and batch size=128)	0.3
non-IID(epoch=5 and batch size=128)	0.3
non-IID(epoch=10 and batch size=128)	0.3

Table III.3. Evaluating the difference in the accuracy when the non-IIDness of the data is varied

non-IIDness	**Epoch and batch size**	**Accuracy**
IID	epoch=1, batch size=64	82.8
non-IID(50 percent)	epoch=1, batch size=64	81.0
non-IID(60 percent)	epoch=1, batch size=64	80.4
non-IID(70 percent)	epoch=1, batch size=64	74.3
non-IID(80 percent)	epoch=1, batch size=64	65.2

and accuracy levels are very low.

In TABLE III.3, Evaluating the difference in the accuracy when the non-IIDness of the data is varied.

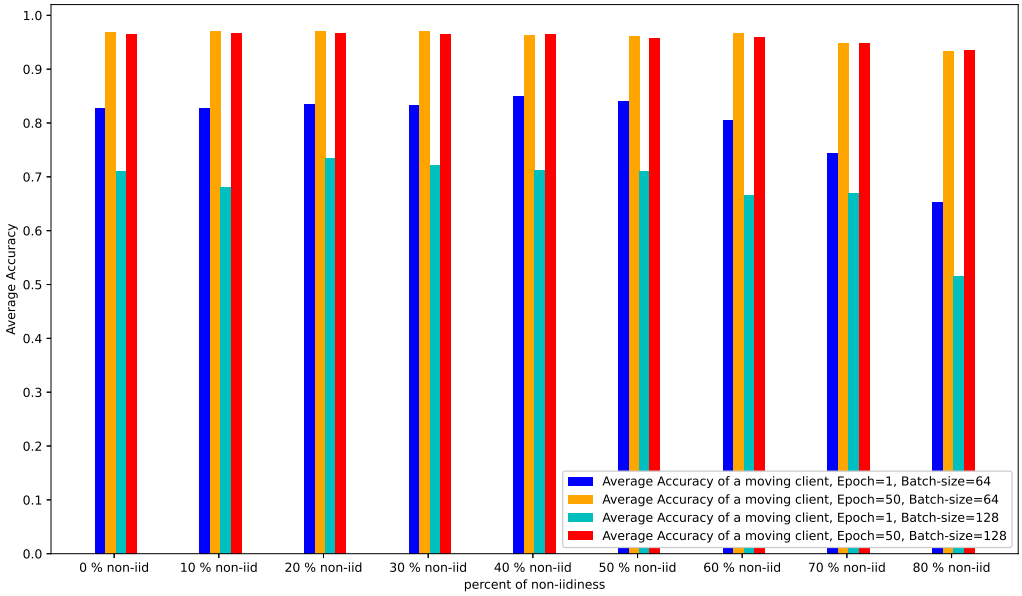

Figure III.12. Comparing the average accuracy when the non-IIDness of the data is varied

CHAPTER IV

IMPROVING FEDERATED LEARNING WITH NON-IID DATA

As with every other technology there is always some pros and some cons attached. Federated Learning being boon in resolving lot of privacy and sensitivity issues related to data sharing among various commodities it also brings some challenges along with itself that needs to be addressed to make this branch of learning more successful and fruitful. Some of the important issues in FL training are highlighted below:

1. **Noise and accuracy:** To protect the data we add noise using the deferential privacy. This definitely helps in enhancing the privacy of the data training but if we find a very considerable amount of deviation that the accuracy graph shows due to noise, we need to reduce the noise level. There needs to be a trade off between the amount of noise being added and the accuracy with which the model performs. [15]

2. **Heterogeneous Systems and Heterogeneous Statistical data:** The Federated Learning models are trained on heterogeneous devices. It is important to ensure federated learning scale effectively on all devices regardless of the type of devices. Also the type of data that is used during the training process is from the different background and may contain data of different type. This dissimilarity must be taken into consideration while training the model for an effective model.

3. **Communication bottlenecks:** This plays an important role and also one the biggest challenges in FL training on large data sets. Communication between the models and also between the global and local model must be reduced as

much as possible to keep the cost very low so that it does not effect cripple the Federated training process. there has been many developments in this field to avoid the communications one such filed is know as associate training. This branch of machine learning reduces the communication between the local model and global model. Concentration is more on peer to peer local model communication so that every other model can advantage of learning process. However, there are some other techniques like dropping stragglers (devices that failed to compute training within the specified time-window). Some other techniques like and model compression and quantization to reduce the bandwidth cost.

4. **Data and Model Poisoning:** We are dealing with client server architecture in the process of federated learning so it is obvious to expect a malicious attacks from third party on the training process in order to destabilize the training itself or make the training process completely useless. It normally come in two different forms Data Poisoning and Model poisoning. There are multiple clients that participate in the learning process on an wireless network, so it is difficult to prevent malicious attackers from sending fake data to training process. This will produce wrong results of unrealistic accuracy graph. Model pointing is way of tampering the models with gradient parameters during training process. The model would then send the same content to global serve as update for aggregation. This would make the Federated training process of no use or may be of little use. [16]

5. **Efficiency and privacy:** We already know that there is a huge challenge lies in the transfer of data between the local training models and centralized global model for update and aggregation. To over come this challenge data/parameters from the federated training process are normally encrypted using some encryption algorithm. This comes with additional cost in terms of resources and time. This

would delay the process of information transfer. Normally we use Secure Multi-Computation for the encryption of the data. we use the Differential Privacy to boost the privacy protection capability in Federated learning. Combining both this process there will be big drop in the accuracy of the model. Therefore finding a suitable trade-off between encryption and privacy protection is big challenge in FL.

6. **Data Partition:** The real world data is very unpredictable and full of noise. In order for a model to be able to predict, it needs to be robust and has to be consistent. To make the training process for the model to be robust, we have designed an algorithm to divide the data-set randomly and provided a non-IID data for training. We came across the scenario that whenever the proportion of IID/non-IID data is varied, there is a change/drop in the accuracy. Our goal through this work is to improve the accuracy for different combination of IID and non-IID data and changes in the algorithm to smooth-en the curve to reduce the gap between IID and non-IID accuracy curve. Our work also takes the moving model clients into consideration. The data on which the model client trains keeps changing as the client keeps moving. In this scenario also we observe a drop in accuracy when the client moves from one data zone to another. Our primary focus is to reduce this accuracy oscillations and smooth-en the curve so that any moving client model will be benefited and produce a smooth accuracy curve. Since we are dealing in server client data transfers, security plays an important role. We would like to pursue the security and privacy of data discussion for future work.

IV.1. Improving Efficiency and Effectiveness

In our experiments, We have identified that the accuracy of this federated model is dependent on lot of factors like epoch, batch size, training algorithm etc. We tried to attenuate all these value with different variations to improvise the

accuracy of the model in an non-IID data zone. As per our observations the significance of the model training with respect to epoch and batch size is smaller than the update in momentum parameter for the SGD model to improvise server side training. Here. we propose a method to improve the performance of the Federated model in non-IID environment utilizing momentum. In this case the Global model accumulates the parameter as usual and then fine tunes the resulting model for the next iteration based on the the momentum and not letting the graph deviate from its original trajectory. This would make sure the local model to have a better updated parameters from global model.

IV.2. Federated Optimization: Methods

We have designed the federated learning methods to handle multiple user devices in collecting data and a global model to coordinate the global learning process in order to take advantage of the federated model. Although there are a lot of concerns related to data band width, network security, privacy for this kind of training Our target is primarily to improvise the accuracy and minimize the training loss to greater extent.

In our experiment we are utilizing Stochastic Gradient Descent to optimize the accuracy graph. Initially the model is initialized with some random values, model calculates the SGD and then the parameters are updated with the new values. In this process the SGD plays a very important role in shaping up the accuracy graph.

Stochastic gradient descent (SGD): It is multi-layer iterative process used for optimizing a functional graph utilizing the smoothness properties. Some of the examples of smoothness properties are differentiable or sub differentiable. Since it replaces the old gradient descent method it is also know as stochastic approximation

of the gradient descent. The gradient descent is calculated using the entire data set which was kind of repetitive task. Stochastic Gradient Descent is calculated only using the subset of the data. After the co-efficient is calculated, all the other gradients is calculated using the same coefficient. This reduces the high computational burden lot of times, especially in very high-dimensional optimization problems. The iterations are pretty fast and also the convergence rate of the graph is much faster when compared to normal traditional way of gradient calculation [17]

$$\theta_i^{'} = \theta_i - \alpha\delta\theta_i \tag{IV.1}$$

where, θ parameter which we want to update. α learning rate. $\frac{\delta L}{\delta\theta}$ gradient of L loss function to minimise, w.r.t. to θ.

Momentum of the curve plays a very significant role in improvising convergence of the graph by making gradient descent robust to certain deviating elements in loss surfaces, e.g., it can reduce the oscillations effect in the areas where the graph id slowly tending away from the actual trajectory like in valleys of the graph. If we consider this property of the momentum we believe and hope that data zone with non-IID data will also will have same effect. To adapt momentum in FL, we keep running averages of aggregated models. So the updated rules of FL along with the momentum can be written as Fed Avg [18] can be written as,

$$v_{r+1} = \beta v_r + \sum_{kS_r} n_k.\Delta_{kr} \sum_{kS_r} n_k \tag{IV.2}$$

$$\theta_{r+1} = v_{r+1} + \mu\theta_r \tag{IV.3}$$

where β is servers momentum constant and $v_0 = 0$

Instead of depending only on the current gradient to update the weight, we update gradient descent with momentum m (momentum), which is an aggregate of gradients. [19] This aggregate of gradients moves exponentially using the current and past gradients (i.e. up to time t).

$$\theta_{t+1} = \theta_t - \alpha m_t \tag{IV.4}$$

$$m_t = \beta m_{t-1} + (1 - \beta)\delta L/\delta\theta_t \tag{IV.5}$$

and m initialised to 0.

Common default value:

$\beta = 0.9$

IV.2.1. Experiment with Updated Training Module and Results

Now that we have seen the accuracy drop in the non-IID environment and we hope to minimise the accuracy drop effect of the model client when moving from one set of data to another set of data using momentum parameter in SGD calculation. In this experiment we have tried to train the client models using the momentum parameters and we have definitely see some improvements in the accuracy.

Plotting a graph of average accuracy of 50 runs(50 times with same epoch and batch size) with Epoch 1, batch size =64. With and without momentum factor. I have considered momentum 0.1,0.5 and 0.7. We are presenting results with momentum 0.7

We see that the graph with the momentum is lot more smoother and the

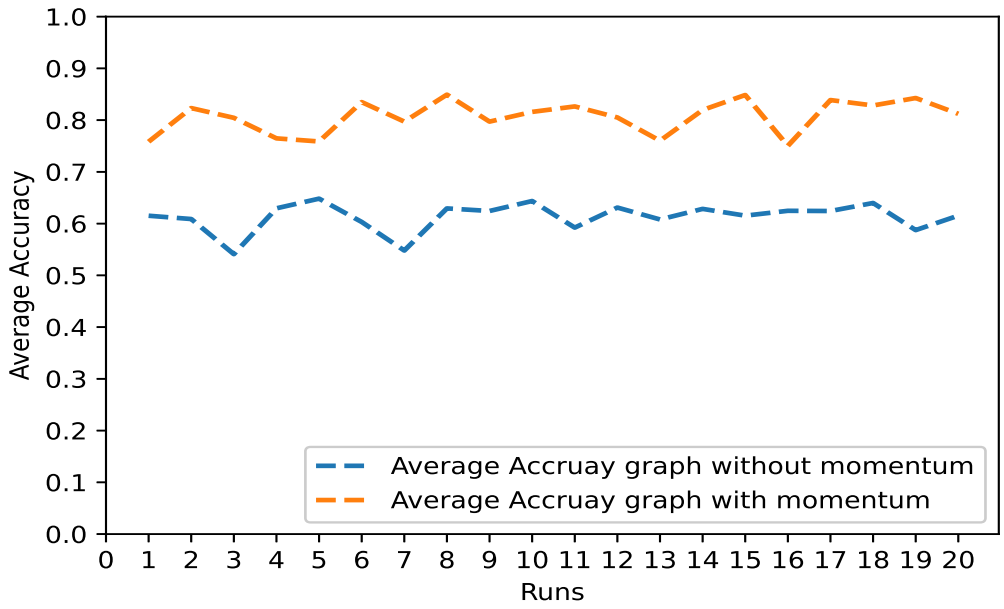

Figure IV.1. 50 runs of average accuracy graph with momentum and without momentum added to the server side training. Epoch=1 and batch size=64

accuracy graph for all the rounds in the training is maintained at higher level. Both the graphs are plotted for same set of data. For Lower epoch the number of times the model client trained is less so the difference in both of these graphs are little higher. We know that with the increase in epoch the accuracy of the model client training increases. It is same case when the momentum is added.

Plotting a graph of average accuracy of 50 runs (Epoch 5, Batch size =64). With and without momentum factor

As mentioned above the average accuracy of the graph increases in with and without momentum. so effective increase of average accuracy between these two types of model training is lesser than the previous one. If we go by this trend we are

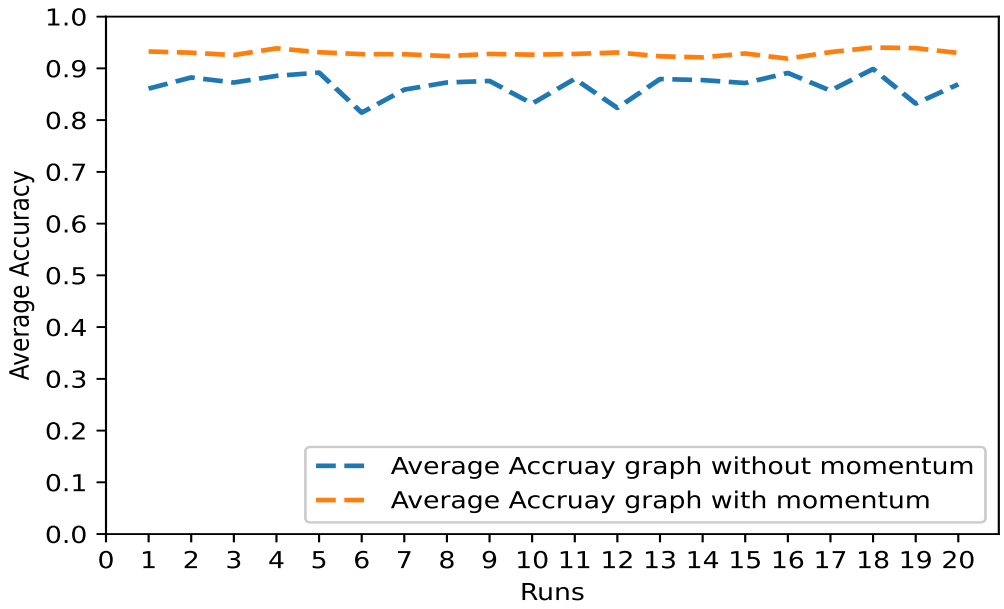

Figure IV.2. 50 runs of average accuracy graph with momentum and without momentum added to the server side training for epoch=5 and batch size=64

expected to have even lesser gap when the epoch level is increased further.

Plotting a graph of average accuracy of 50 runs (Epcoh 10, batch size =64). With and without momentum factor.

On an average we see 10-15 percent of increase in the accuracy when the model client is trained with the momentum. The momentum component in the SGD sees that the training graph does not deviate from its trajectory thus smoothing the training graph. This makes sure that the graph accuracy is better when trained with momentum. We added the accuracy runs data in the below report. We can compare the accuracy improvement in all runs either with and without momentum Accuracy when the epochs and batch size is constant.

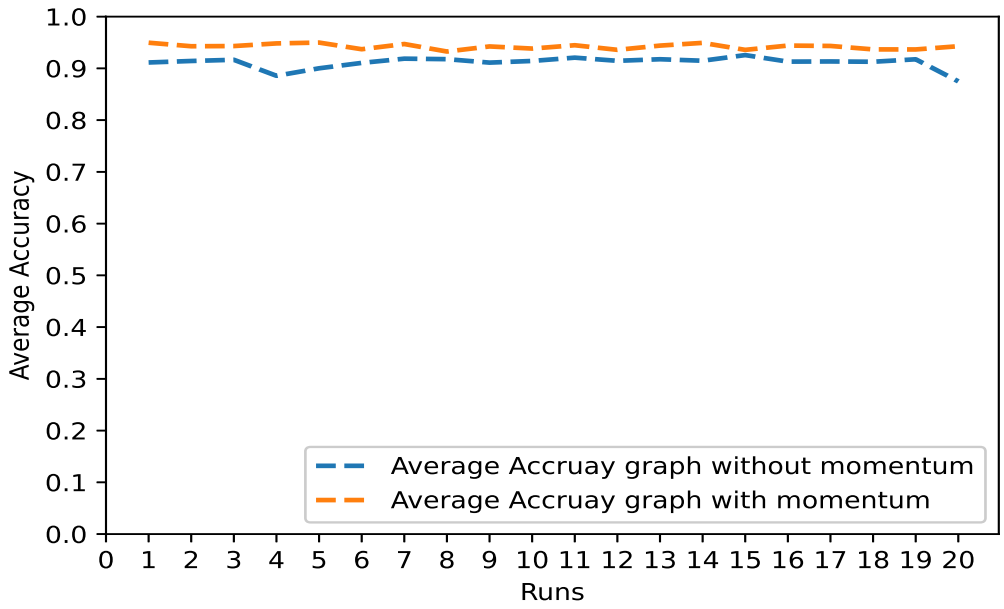

Figure IV.3. 50 runs of average accuracy graph with momentum and without momentum added to the server side training for epoch=10 and batch size=64

IV.2.2. Results

In TABLE IV.1, we are showing the accuracy of the model client trained after the momentum parameter is added towards the server side training. We have considered almost similar epochs and batch sizes to show the deference form the previous training.

In TABLE IV.1: we are presenting results of basic data zone settings i.e. very ideal conditions of IID data training. We also presenting data when the FL model trains on non-IID data with different variation in epoch and batch size. Accuracy of the Federated Learning model is very high when the data distribution is of IID. It is almost same with and without momentum, there was not much of a difference. However, when the FL model trains with data in non-IID distributed fashion and

Table IV.1. Analysis of Average Accuracy of 50 runs of model client training with different epoch and batch size

Data-zone	epoch and batch size	Average Accuracy
IID	epoch=1 and batch size=64	94.7
non-IID(80 percent non-IIDness)	epoch=1 and batch size=64	81.0
non-IID(80 percent non-IIDness)	epoch=5 and batch size=64	85.8
non-IID(80 percent non-IIDness)	epoch=10 and batch size=64	91.6
IID	epoch=1 and batch size=128	94.7
non-IID(80 percent non-IIDness)	epoch=1 and batch size=128	77.2
non-IID(80 percent non-IIDness)	epoch=5 and batch size=128	84.4
non-IID(80 percent non-IIDness)	epoch=10 and batch size=128	89.7

Table IV.2. Evaluating the average loos of the accuracy during the model client training

80 percent non-IIDness	average Loss for 50 rounds
IID (epoch=10 and batch size=64)	0.1
non-IID(epoch=1 and batch size=64)	0.2
non-IID(epoch=5 and batch size=64)	0.2
non-IID(epoch=10 and batch size=64)	0.2
non-IID(epoch=1 and batch size=128)	0.3
non-IID(epoch=5 and batch size=128)	0.3
non-IID(epoch=10 and batch size=128)	0.3

epoch as 1 or 5 or 10 , we had lower Average accuracy without momentum, but now we see the accuracy is higher. The difference between the IID and non-IID accuracy was almost a drop of more than 25 percent when the model clients are moving. After the momentum considerations the accuracy drop has reduces to 13 percent.

In TABLE IV.2, we are presenting the average loss of the accuracy during client training. We know that the training loss is inversely proportional to the accuracy of Federated Learning model. With that expectation we hope to prove that the average loss reduces with the update in momentum parameter.

In TABLE IV.2 the average Loss of the Fl training model decreases with the increase in epoch. We also observed that when the model is trained with momentum parameter there is further drop in the loss during training. This is good sign for the

Table IV.3. Comparing the Average Accuracy of the graph with and without momentum

Data-zone	Avg. Accu. W/o M	Avg.Accu. with M
non-IID(epoch=1, batch size=64)	61.5	81.0
non-IID(epoch=1, batch size=128)	58.5	77.2
non-IID(epoch=10, batch size=64)	82.2	91.6
non-IID(epoch=10, batch size=128)	81.8	89.7

training and this makes sure that the Federated Learning model has improved.

In TABLE IV.3: we present the comparison of average accuracy for a federated model that is trained with different epochs and batch sizes. Also it is important to note that the FL model moves in between the data zones during training. So the result accumulates the final improvised accuracy with the initial accuracy for same parameters.

In TABLE IV.3: we can observe that the addition of momentum parameter to the SGD training has brought lot of accuracy improvement in the Federated Learning model. The drop that we saw when the data was updated from IID to non IID is also minimised for an epoch=1 the accuracy dropped to 64 percent. We also saw lot of sharp drop in the accuracy graph whenever the model moved to next data zone. The accuracy rate has improved over 81 percent which is almost improvisation of 19 percent. As we test the same scenarios with other epochs we find that the improvisation is of 10 percent. This improvisation is similar to other setup training as well. We also tried adding different momentum constants to improvise the model further. We see that accuracy rate increases in all the cases. The effect of using moments on server side training improvises the accuracy to large extent.

CHAPTER V

CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we have focused on the behavior of the Federated Learning if the data is non identical unevenly distributed among the client model for training. We also focused on the local model clients that are not stagnate or fixed to one data set of data for training. It also mean that the models are roaming in between the different sets of non-IID data sets which is derived from MNIST data set.

During the course of this work, we found that when the data is very unevenly distributed and models are moving around the accuracy of the model training goes down by almost 25 percent. Our main focus in this study was to design a solution for such federated models so that the accuracy drop is not very high and all the moving clients can also take advantage of this solution to get a better and smoother accuracy graph. We also identified that some parameters like Epoch, Batch size of the training data, learning rate of the algorithm plays a very vital role in determining accuracy of a federated model. We experimented with these parameters and identified that the increase in epoch also increases the accuracy of the FL model. Increase in the batch size has the opposite effect. Higher the batch size lower were the results of the FL model, while all the other parameters including training data was same. These feature are not rigorous and works by trial and error method so we needed to identify something else that had the same effect but that can be tuned during the course of training time.

During the course of model training, we found out that the technique that we are using for optimization Stochastic Gradient Descent can be tuned further

with the help of momentum. It is kind of an added feature to the gradient descent optimization algorithm so that the direction of the search does not alter during the training process and the training model can over come some noisy gradients that form huge oscillations in the graph. Working on this SGD feature we added to the FL model training and performed the same experiment with IID and non-IID data setup. We found that the there was at least increase of 10-15 percent of average graph accuracy with momentum. we experiment this with the moving local models that had huge accuracy drop when it moved from one data zone to another. On the similar line of non-IID data we found that the moving clients were also at advantage during training due to momentum. One of the aspects of the FL model training that is not covered in this study is privacy and bandwidth requirement of such training in FL model. In a future work, we hope to expand our experimental setting by testing different models, data sets, and hyper parameters. We also want to study the increase in computational activities due to addition of momentum to the FL model along with bandwidth requirement for this experiment in real world scenarios.

BIBLIOGRAPHY

[1] Canh Dinh, Nguyen Tran, Minh Nguyen, Choong Hong, Wei Bao, Albert Zomaya and Vincent Gramoli , "Federated learning over wireless networks: Convergence analysis and resource allocation," *arxiv preprint arXiv:1910.13067*, pp. 2–3, 2019. [Online]. Available: https://arxiv.org/pdf/1910.13067.pdf

[2] Q. L. Quan Chen, Bingsheng He and Y. Diao, "Federated learning on non-iid data silos: An experimental study," *arxiv preprint arXiv:2102.02079*, 2021. [Online]. Available: https://arxiv.org/pdf/2102.02079.pdf

[3] M. Najafabadi, F. Villanustre, T. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, pp. 1–2, 2015.

[4] Qiang Yang, Tianjian Chen, Yongxin Tong and Yang Liu, "Federated machine learning: Concept and applications," *arxiv preprint arXiv:1902.04885*, 2019. [Online]. Available: https://arxiv.org/pdf/1902.04885.pdf

[5] T. Alam and R. Gupta, "Federated learning and its role in the privacy preservation of iot devices," *Future Internet*, vol. 14, no. 9, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/9/246

[6] A. Nguyen, T. Do, M. Tran, B. X. Nguyen, C. Duong, T. Phan, E. Tjiputra, and Q. D. Tran, "Deep federated learning for autonomous driving," 2021. [Online]. Available: https://arxiv.org/abs/2110.05754

[7] Nicola Rieke, Jonny Hancox and Wenqi Li, "The future of digital health with federated learning," *arxiv preprint arXiv:2003.08119*, 2014. [Online]. Available: https://arxiv.org/pdf/2003.08119.pdf

[8] Prayitno, C.-R. Shyu, K. T. Putra, H.-C. Chen, Y.-Y. Tsai, K. S. M. T. Hossain, W. Jiang, and Z.-Y. Shae, "A systematic review of federated learning in the healthcare area: From the perspective of data properties and applications," *Applied Sciences*, vol. 11, no. 23, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/23/11191

[9] Brendan McMahan and Abhradeep Thakurta, "Federated learning with formal differential privacy guarantees," *Google Research*, 2022. [Online]. Available: https://ai.googleblog.com/2022/02/federated-learning-with-formal.html

[10] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[11] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, 2021.

[12] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[13] Y. Bengio, Y. Lecun, and G. Hinton, "Deep learning for ai," *Communications of the ACM*, vol. 64, no. 7, pp. 58–65, 2021.

[14] K. Patel, "Mnist handwritten digits classification using a convolutions neural network," 2019.

[15] G. M. Jain Priyank and N. Khare, "Differential privacy: its technological prescriptive using big data," 2018. [Online]. Available: https://doi.org/10.1186/s40537-018-0124-9

[16] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *Computer Security – ESORICS 2020*, L. Chen, N. Li, K. Liang, and S. Schneider, Eds. Cham: Springer International Publishing, 2020, pp. 480–501.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[18] S. Jelassi and Y. Li, "Towards understanding how momentum improves generalization in deep learning," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 9965–10 040. [Online]. Available: https://proceedings.mlr.press/v162/jelassi22a.html

[19] Y. Liu, Y. Gao, and W. Yin, "An improved analysis of stochastic gradient descent with momentum," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 18 261–18 271. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/d3f5d4de09ea19461dab00590df91e4f-Paper.pdf

APPENDIX A

SELECTED CODE SNIPPETS

We have used the python code for the all the development activities. Python has some of the most simplified libraries that helped us lot in the development. some important ones that we used for experiments are torch, torch vision, numpy, arg parse matplotlib etc.

A.0.1. Data Partition

We designed multiple ways to partition the data.

a. partition == "IID-data":

b. partition == "noniid-data":

c. partition == "variable-percentage-non-IID-data":

d. partition == "IID-non-IID-mixed":

```
elif partition == "noniid-data":
        percentage_of_non_iid= n_percent

        contain=[]
        for i in range(n_parties):
            current=[0,1,2,3,4,5,6,7,8,9]
            contain.append(current)
        #print("contain",contain)
        train_data ={i:np.ndarray(0,dtype=np.int64) for i in range(n_parties)}
        test_data ={i:np.ndarray(0,dtype=np.int64) for i in range(n_parties)}
        count=0
        for i in range(n_parties):
            idx_k = np.where(y_train==i)[0]
            from datetime import datetime
            np.random.seed((int(datetime.utcnow().timestamp())))
            np.random.shuffle(idx_k)
            if(count==i):
                split = np.array_split(idx_k,2)
                ids=0
                train_data[i]=np.append(train_data[i],split[1])

            count+=1
        for i in range(n_parties):
            idx_t = np.where(y_test==i)[0]
            np.random.shuffle(idx_t)
            split = np.array_split(idx_t,n_parties)
            ids=0
            for t in range(n_parties):
                if i in contain[t]:
                    test_data[t]=np.append(test_data[t],split[ids])
                    ids+=1
```

Figure A.1. Partitioning the MNIST training set and test set. Training into non-IID data and testing as IID data

A.0.2. local model training

```
def training(Accuracy_Matrix, nets, selected, global_model, args, train_data,␣
↪test_data, test_dl = None, device="cpu"):
    avg_acc = 0.0

    a_list = []
    d_list = []
    n_list = []
    global_model.to(device)
    for net_id, net in nets.items():
        if net_id not in selected:
            continue
        train_data_ids = train_data[net_id]
        test_data_ids=test_data[net_id]
        logger.info("\n")
        logger.info("Client %s. No_of_training_samples: %d" % (str(net_id),␣
↪len(train_data_ids)))
        logger.info("\n")
        logger.info("Client %s. No_of_testing_samples: %d" % (str(net_id),␣
↪len(test_data_ids)))
        net.to(device)


        train_local, test_local, _, _ = get_dataloader(args.dataset, args.
↪datadir, args.batch_size, 32, train_data_ids, noise_level, net_id, args.
↪n_parties-1)
        train_global, test_global, _, _ = get_dataloader(args.dataset, args.
↪datadir, args.batch_size, 32)
        n_epoch = args.epochs
        trainacc, testacc, a_i, d_i = local_training(net_id, net, global_model,␣
↪train_local, test_dl, n_epoch, args.lr, args.optimizer, device=device)
        Accuracy_Matrix[net_id].append(testacc)
        a_list.append(a_i)
        d_list.append(d_i)
        n_i = len(train_dl_local)
        n_list.append(n_i)
        logger.info("Client %d Final Test Accuracy %f" % (net_id, testacc))
        avg_acc += testacc
```

Figure A.2. Initializing all the model client for training.

A.0.3. Simple convolution model

```
optimizer = optim.SGD(filter(lambda p: p.requires_grad, net.parameters()),␣
 ↪lr=lr, momentum=args.rho, weight_decay=args.reg)
criterion = nn.CrossEntropyLoss().to(device)
```

Figure A.3. SGD optimizer for Gradient with momentum parameter

A.0.4. Loss function

```
   tau = 0
    round_loss=0
    for epoch in range(epochs):
        epoch_loss_collector = []
        for tmp in train_dataloader:
            for batch_idx, (x, target) in enumerate(tmp):
                x, target = x.to(device), target.to(device)
                optimizer.zero_grad()
                x.requires_grad = True
                target.requires_grad = False
                target = target.long()
                out = net(x)
                loss = criterion(out, target)
                loss.backward()
                optimizer.step()
                tau = tau + 1
                epoch_loss_collector.append(loss.item())
        epoch_loss = sum(epoch_loss_collector) / len(epoch_loss_collector)
        logger.info('Epoch: %d Loss: %f' % (epoch, epoch_loss))
        round_loss+=epoch_loss
    print("loss in this round", round_loss)
```

Figure A.4. Calculating the training loss per round