Proxy Data Based Federated Learning Using GMMs

submitted in partial fulfillment of the requirements

for the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

ANIKET KUMAR CS20M002

Supervisor(s)

Dr. Kalidas Yeturu

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

May 2022

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/-data/fact/source in my submission to the best of my knowledge. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Tirupati
Date: 30-05-2022

Signature
ANIKET KUMAR
CS20M002

BONA FIDE CERTIFICATE

This is to certify that the report titled **Proxy Data Based Federated Learning Using GMMs**, submitted by **ANIKET KUMAR**, to the Indian Institute of Technology, Tirupati, for the award of the degree of **Master of Technology**, is a bona fide record of the project work done by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Tirupati
Date: 30-05-2022

Dr. Kalidas Yeturu
Guide
Assistant Professor
Department of Computer Science & Engineering
IIT Tirupati - 517501

ACKNOWLEDGMENTS

I would like to express my special thanks of gratitude to **Dr. Kalidas Yeturu** for his able guidance and support in completing my project. He encouraged me to work on this wonderful idea which aided me in conducting extensive study and learning about a bunch of new topics in distributed learning. I am thankful to all the Department of Computer Science and Engineering faculty members at IIT Tirupati for their continuous support throughout this project and all the courses here at IIT Tirupati and making this memorable, successful journey. And a lifelong learning experience.

I also admire the constant help and guidance of the Ericsson Research team **Swarup Mohalik, Ankit Jauhari, Anil Ramachandran Nair** for providing me with all the facility that was required.

I would also like to give special thanks to **Aditi Palit** for her long hours of brainstorming and code reviews.

At last, I ended up thanking all who helped me finalize the project within the limited time frame.

ABSTRACT

KEYWORDS: Federated Learning; GMM; Proxy-data; FedAvg; IID; Non-IID

Traditionally, all machine learning model training requires a central server where training data is stored, and the model is trained on the data. But, this approach has a downside. Firstly, there is a single point of failure when the central entity goes offline. Secondly, there is a privacy concern for all those users participating in the process of data gathering. Federated learning is a new form of machine learning technique which allows a central model to learn from shared local model while keeping all the training data on local machine itself.[4]. A local machine can be an edge device like a mobile phone, watch, etc. These days data and privacy is a prime concern. Using these techniques, we can ensure privacy and still achieve good accuracy.

This report will compare two verticals of achieving federated learning, proxy-data-based federated learning and weights update using weighted averaging. We replicated all the experiments of the state-of-the-art FedAvg algorithm paper as described in the article Communication-Efficient Learning of Deep Networks from Decentralized Data.[7]. And also demonstrate the effective implementation of the Gaussian Mixture Model (GMM) with the Expectation-Maximization (EM) algorithm according to the vanilla federated learning paradigm. We build a baseline GMM and a federated implementation of the same model to compare their performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

API	Application program interface
EM	Expectation-Maximization
GDPR	General Data Protection Regulation
GMM	Gaussian mixture model
SGD	Stochastic Gradient Descent
VP	Vertically partitioned

CHAPTER 1

INTRODUCTION

With an increase in phones and gadgets as the primary computing devices in daily activity for many people.[8; 2]. The robust hardware detectors on these devices (including ai cameras, microphones, GPS, etc.) added that these are on-the-go devices. One can carry it everywhere, meaning they have access to massive private data. So, a model trained on such data can provide a better user experience by powering more intelligent applications.[4]. Since the data is confidential and private, there are risks and responsibilities for holding it in some centralized location.

There is already a machine learning technique that allows users to learn a model without disclosing their data; collaboratively. This technique is termed federated learning.[7]. Since the teaching is the happened by an open federation of participating devices (we call them local clients) coordinated by a client-server with an appropriate load. Each client has a local dataset (which can be constant or variable in every iteration) that is never revealed to a central server. Instead of sending the local dataset to central server, each client trained the current global model maintained by the central sever and only update like parameters or weights are communicated. This adheres to the application of the principle of focused collection or data minimization proposed by the 2012 White House report on the privacy of consumer data.[3] Since these parameters or weights are specific to improve the central model, so once it is applied, there is no need to store it.

The main advantage of federated learning is the decoupling of training the model from the need for direct access to the client's private data. Still, some trust from the coordinating servers is also required. However, for our application, where data privacy is of utmost importance, federated learning can significantly reduce privacy risk by limiting the attack surface to only clients.

Our primary contributions are 1) an implementation of the one-shot learning using GMMs for the generation of proxy data according to the federated learning paradigm and 2) an improvisation of silhouette coefficient (a measure of similarity of a data point is within-cluster (cohesion) compared to other clusters (separation)) 3) an extensive empirical evaluation of the proposed approach. More precisely, we used a vanilla federated learning paradigm. The catch is instead of training a stochastic gradient descent(SGD), we fitted GMMs to the training data and sent the parameters to a central server for sampling. We perform extensive experiments on this approach, both on IID and non-IID data distribution.

1.1 Motivation

The motivation for this approach is data preserving data privacy and achieving approximate accuracy as centralized machine learning settings. With increasing data breaches splashed across front-page news, companies have good reason to take security seriously. Federated learning is a powerful technique, yet it amazes without disclosing the data. We have carried out extensive experiments and empirical evaluation of this approach. Moreover, we observe great results, which drive us to further work on this.

1.2 Scope

Federated learning enables collaborative model learning without sharing clients' raw data. This technique, over time, draws attention from industries where data privacy is demanded. Especially in the field where multiparty collaboration is required for modeling applications, the data is owned by multiple (two or more) parties. Each holder has its records of different feature sets with common entities. Such kind of multiparty data setting where each party has some data of a common entity is called vertically partitioned (VP) data.[12].

While a model trained upon an integrated dataset improves the performance, organizations cannot share data due to legal restrictions or competition between participants. For example, an E-commerce company (ShoopingKart), a digital finance company(ATMPe), and a bank(Bank of IIT) collect different information about the same entity. The E-commerce company has all the purchased history of the entity. The bank has customer information, such as account balance. The digital finance company has information like mandate payments, loan repayments, credit scores, etc. Suppose a person submits a loan application to a finance company, and the finance company wants to evaluate the credit risk for that person, so that requires a model trained on all the above three parties' data.

In such scenarios, federated learning models appear popular, and these raise the need for an efficient federated learning algorithm like One-shot GMMs for a variety of data distribution. As direct access to the data or sharing of the data to other organization is often prohibited due to legal and commercial issues. For legal reason, most countries worldwide have made laws in the protection of data security and privacy. For example, the European Union made the General Data Protection Regulation (GDPR)[1] to protect users' personal privacy and data security.

CHAPTER 2

Background

2.1 Federated Learning

Federated learning is a new technique of machine learning in which a model is trained by combining the updates received from local clients without revealing the local client's data. There are two approaches to central server update:

- By sampling synthetic data using GMMs
- By sending weights to the central and aggregation

There are many algorithm for both approaches. For our work, we are going to demonstrate FedAvg [7] and Gaussian Mixture Model (GMM) with Expectation-Maximization (EM) [6] algorithm to carried out experiments.

2.1.1 The Federated Averaging Algorithm

Federated Averaging is the communication efficient algorithm for distributed training over huge number of clients and on heterogeneous data set.

These are steps in the algorithm :

- Server: Central server initialize a random weight w0. Fig : 2.1
- Server: For every round take m Clients and send initial weight w0. Fig 2.5
- Client: Set weights received by Server and start training on local data. Fig 2.3
- Client : Send updated weights to Server. Fig 2.4
- Server : Do set wi = avg([m clients weight]). Fig 2.4
- Server : Repeat the process but send wi

A typical implementation of federated averaging is given in pseudocode 1. Here, the central and local clients model are of same type.

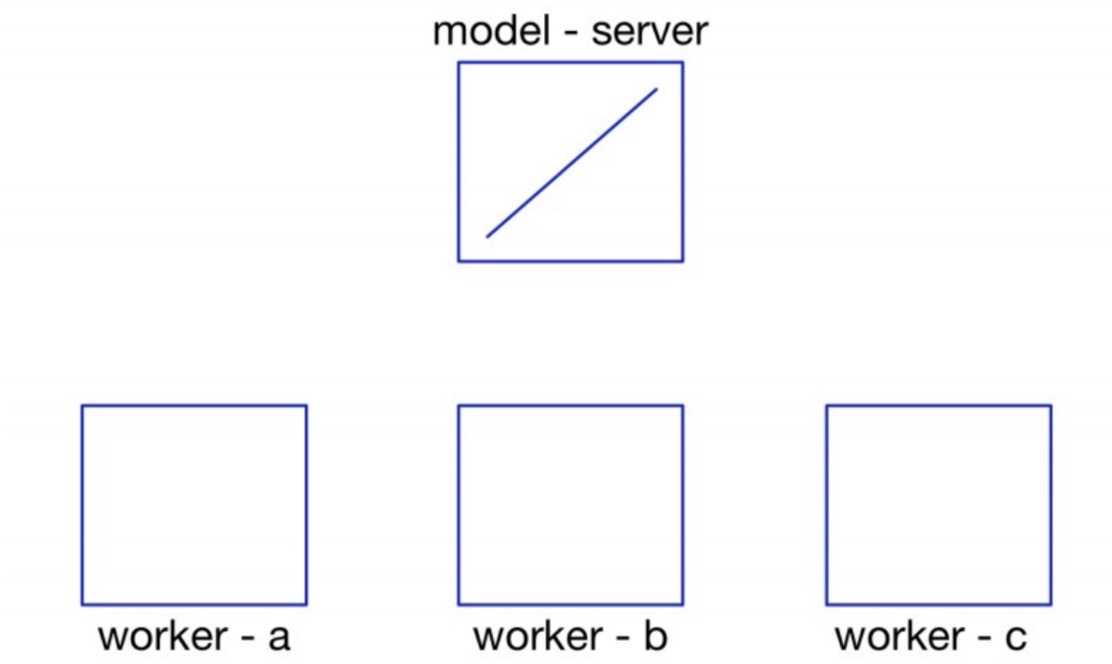

Figure 2.1: A random model is chosen at server

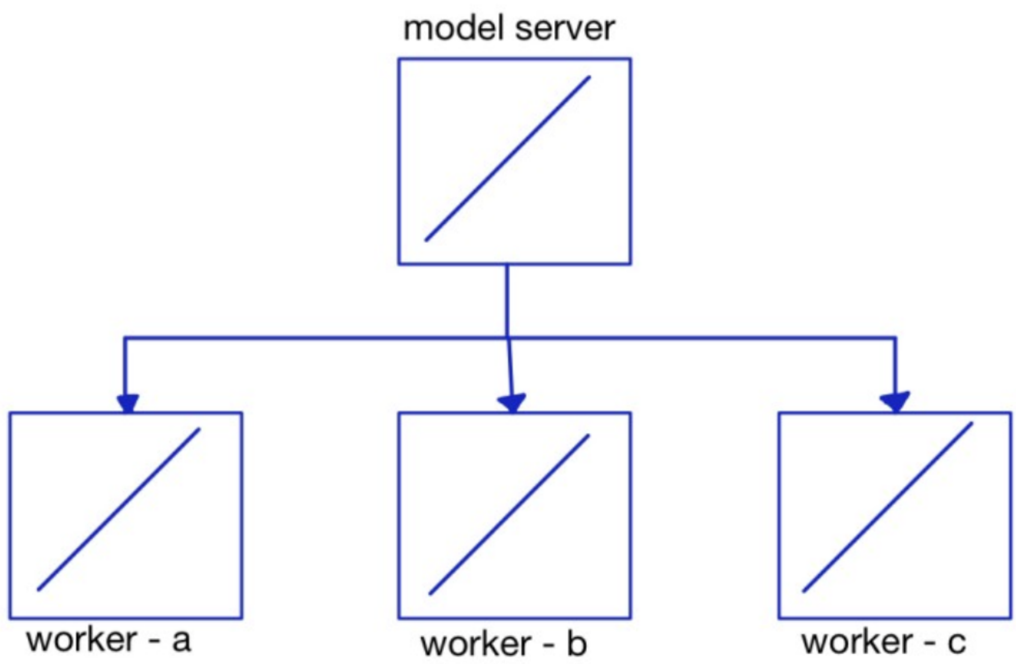

Figure 2.2: Initial model is transmitted to local clients

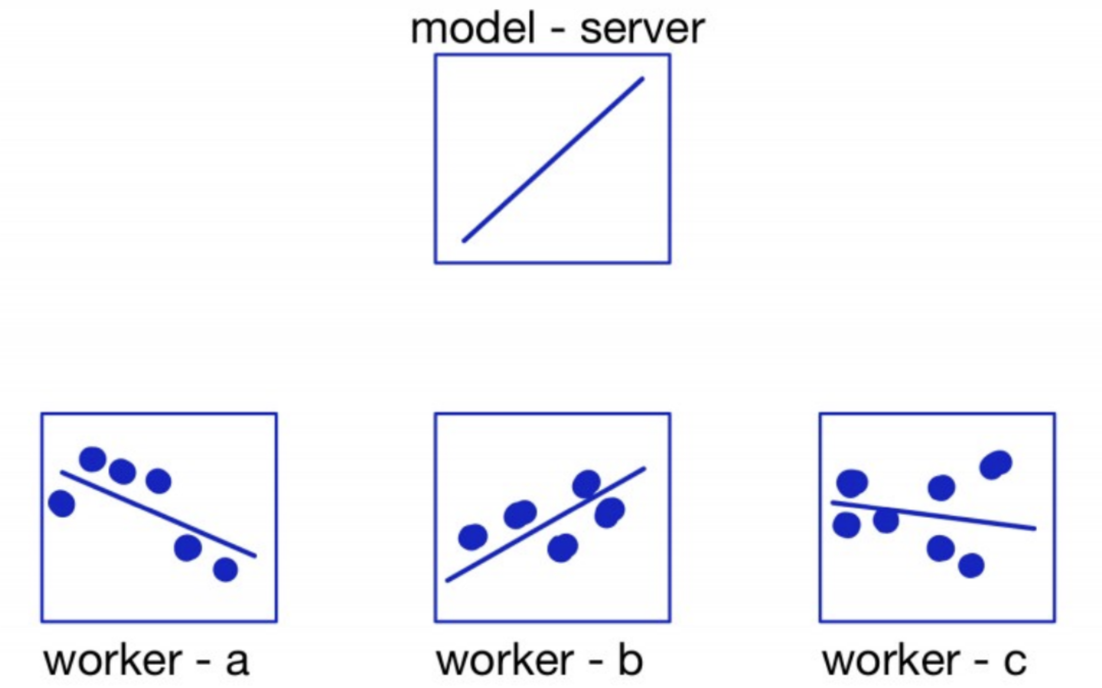

Figure 2.3: Local clients trained the model from its own data

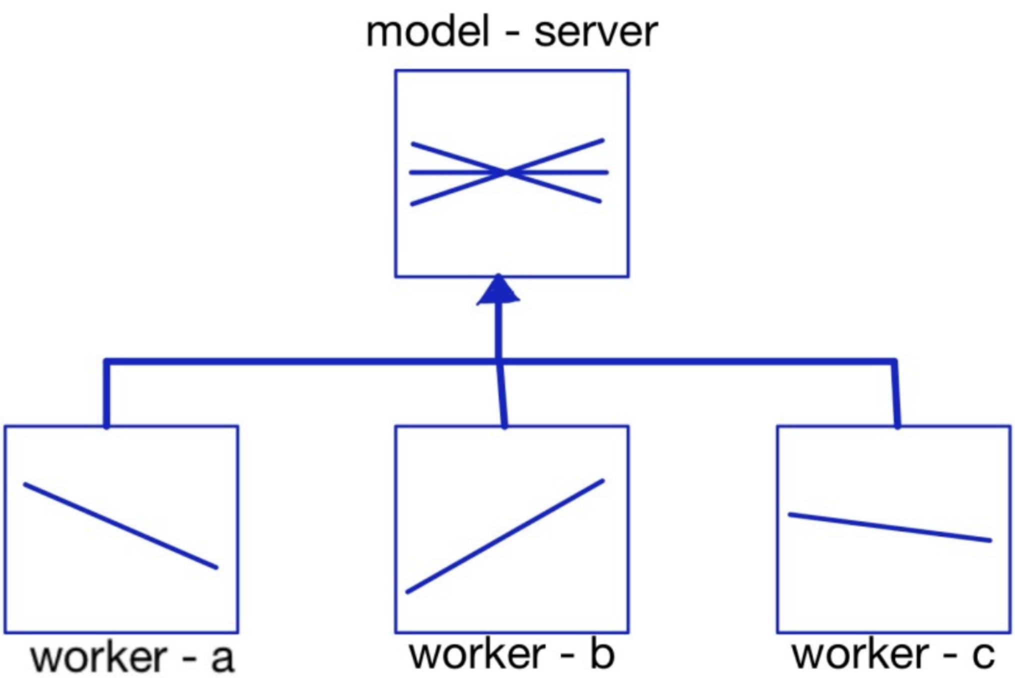

Figure 2.4: Weights from local clients sent to server to aggregate using FedAvg

A slight variation of the federated averaging algorithm is FedSGD, where $C = 1$ i.e. all clients will take participate in each round; $E = 1$ i.e. only single step of gradient descent; $B = \infty$ i.e. the full local dataset is considered as a single mini batch.

A

Pseudocode 1: Federated Averaging Algorithm

```
Function FederatedAverage(K,B,E, η)
    K : number of clients in fraction
    B : mini batch size of SGD
    E : number of epochs
    η : learningrate

    Server :
    C : total number of clients
    k : indexes of K
    w0< −random_weight()
    for i = 1,2,... do
        m <- max(C.K,1)
        S_t <- choose_random(m) //choose random m clients
        for k ∈S_t do
            w^k_t+1, n_k <- clientTraining(k,w_t)
            // store w^k_t+1 in an array
        n = ∑_{k=1}^{K} n_k
        w_t+1 <- ∑_{k=1}^{K} n_k/n ∗ w^k_t+1

        clientTraining(k,w_t):
        for i ∈ 1toE do
            for batch b ∈ B do
                w < −w − η∇l(w;b)
        return w, n_k to server
```

For general non-linear problems, averaging model gives arbitraty results with different initial starting point. For example,

$D = (1,1),(-1,1)$

$f(x) = max(0, w \cdot x)$

$L(f(x),y) = (f(x)y)^2$

Let, D1 = (1,1) and D2 = (-1,1),

On D1, w = 1

On D2, w = -1

Average, w = 0

On D1, max(0,1*0)

On D2, max(0,-1*0)

Error when w = 0, 2 units

Error when w = 1, 1 units

On the combined data set, w = 1 or w = -1 has lesser error

On the combined data set, w = 0 has higher error

that means, averaged weight is having higher error

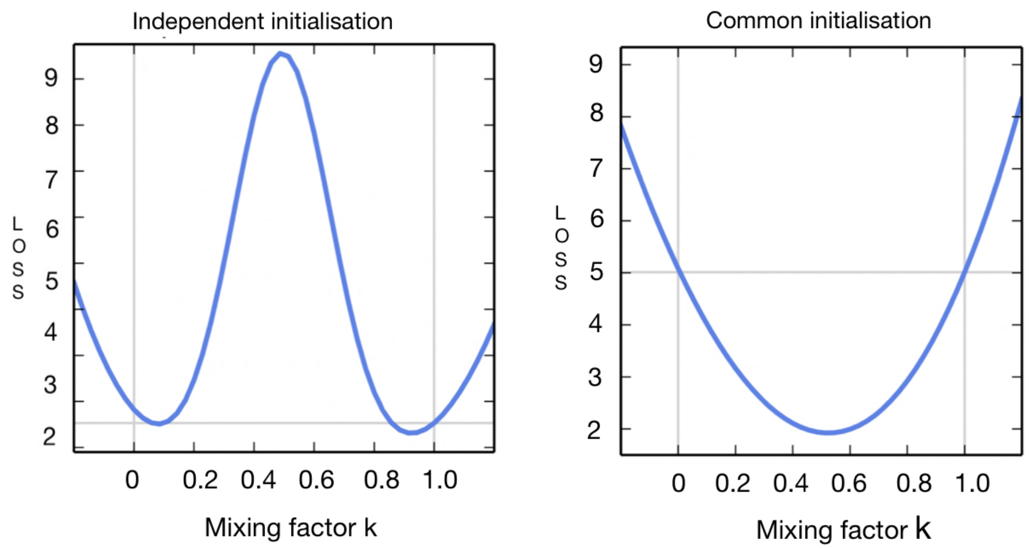

Figure 2.5: The loss value recorded when full CIFAR10 training dataset is trained on two of models and the parameters are averaged using k factor ($k * w_0 + (1-k) * w_1$). The two models are trained using SGD. In the first plot they are initialised by random seed every time while on the second plot they are initialized used the same value. With same initialization, averaging is produced better loss values than random initialization.

2.2 Gaussian mixture models

A GMM is a probabilistic model, which says all the data points from sample space belongs to some mixture of finite number of gaussian/normal distribution with unknown mean and covariance value. Its similar to k-means clustring where clusters gives information about the covariance structure of the data.

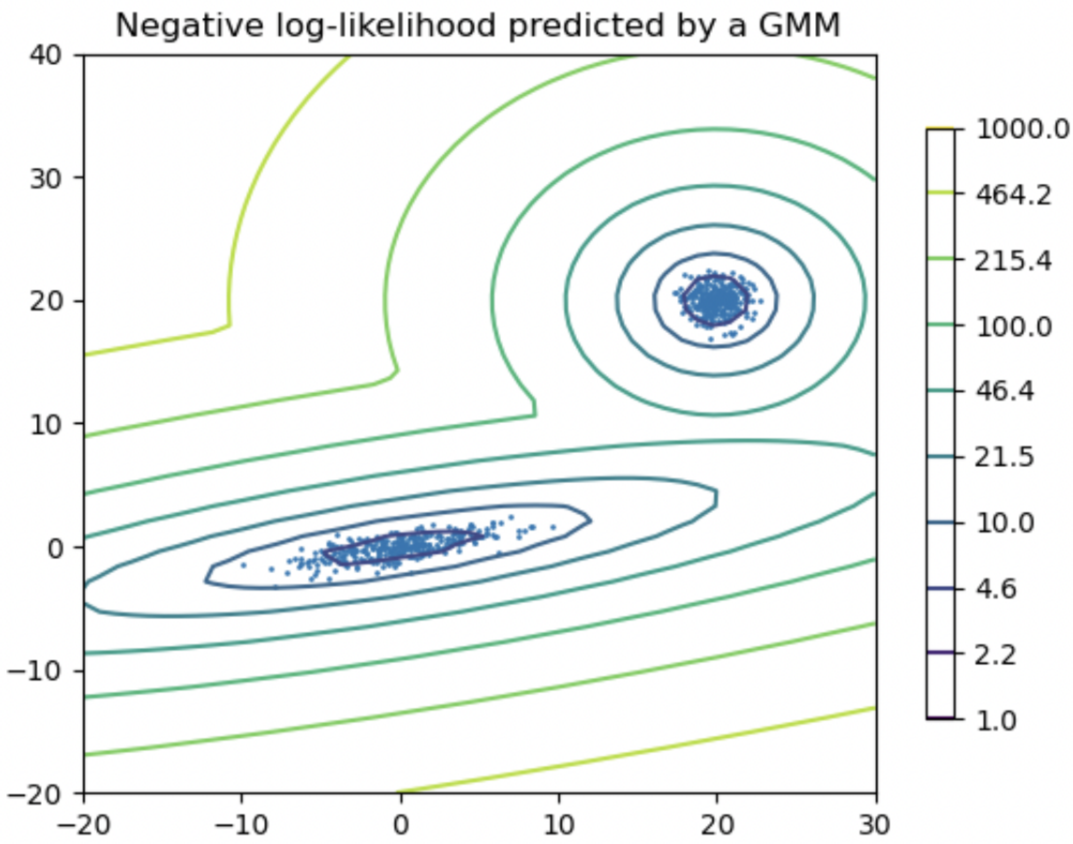

Figure 2.6: Two-component Gaussian mixture model. [10].

A mixture of Gaussian mixture models can be mathematically written as :

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x} \mid \mu k, \sum k\right) \tag{2.1}$$

where, $x \in R^d$,

π_k is the weight of k^{th} Gaussian

$\mu_k \in R^d$ of k^{th} Gaussian

$\sum_k \in R^{d*d} covairancematrix$

$\mathcal{N} \in R^d$ is gaussian which is of the form equation 2.2

$$\mathcal{N}(\mathbf{x} \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\right) \tag{2.2}$$

2.2.1 Expectation-Maximization (EM) algorithm

Expectation-Maximization (EM) algorithm helps in finding optimal gaussian mixture parameter for a given sample space. The main difficulty while fitting a GMM model on unlabeled data is that it is not known that whether data point cannot be directly observed it can be inferred through latent component. If one has access to information about the latent component for each point, it will be easy to build a separate GMM for that point. Expectation-Maximization is well formed statistical method to solve this issue using iterative approach. It takes a random component and for each point compute probability where it will be generated by any component in the model. Then with hyper parameter adjustment one can maximize the likelihood of the data points for component.

Pseudocode 2: Expectation-Maximization (EM) algorithm

Function `ExpectationMaximization`$(\eta, \sum, \tau)$

$\tau = \{\tau_1, \tau_2, ..., \tau_i\}$ *weight of* i^{th}*component*

$\sum : \{\sum_1, \sum_2, ..., \sum_i\}$*covariance matrix of* i^{th}*component*

$\eta : \{\eta_1, \eta_2, ..., \eta_i\}$ *mean of the* i^{th}*component*

Initialise with random $\tau, \sum, \eta$

take E-step and Calculate $T_{k,i}$:

$$T_{k,i} = \frac{p_k\left(x_i; \mu_k, \Sigma_k\right)\tau_k}{\sum_{k=1}^{K} p_k\left(x_i; \mu_k, \Sigma_k\right)\tau_k} \tag{2.3}$$

take M-step and update $\tau, \sum, \eta$:

$$\tau = [\tau_k]\, k = 1 : K = \frac{1}{N}\sum i = 1^N T_{k,i} \forall k \tag{2.4}$$

$$\mu = [\mu_k]\, k = 1 : K = \frac{\sum i = 1^N T_{k,i} x_i}{\sum_{i=1}^{N} T_{k,i}} \forall k \tag{2.5}$$

$$\Sigma^* = [\Sigma_k]\, k = 1 : K = \frac{\sum i = 1^N T_{k,i}\left(x_i - \mu_k\right)\left(x_i - \mu_k\right)^\top}{\sum_{i=1}^{N} T_{k,i}} \tag{2.6}$$

Repeat E-step and M-step alternatively until converge

The flow chart of Expectation-Maximization (EM) algorithm can be seen here.2.7

2.2.2 One-shot Learning using GMMs

Implementation of vanilla federated learning paradigm using GMMs, we named as One-shot learning since there is no iterative exchange of weights or parameters like FedAvg. Here central model and local clients model need not be same. This is due to fact that our learning is done through proxy data which is sampled at central server. As there is no exact weights or parameters which need to set or unset. Hence, there is a decoupling of model architecture of central and clients.

These are steps in the algorithm :

- Server: Central server initialize any random model. 2.1
- Server: Listen for GMMs from m clients. 2.5
- Client: Build GMM on its data. Fig 2.3
- Client : Send GMM parameters to Server. Fig 2.4
- Server : Sample x number of data points build training data. Fig 2.4
- Server : Repeat if more data points required

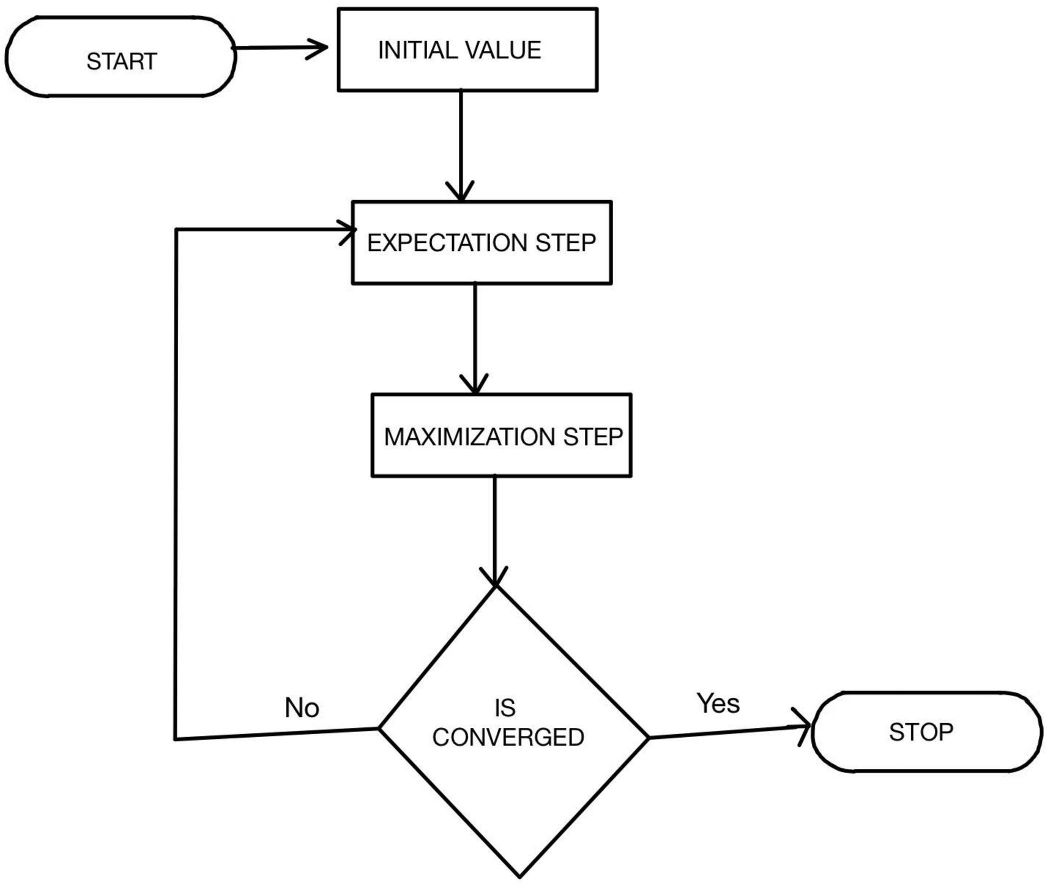

Figure 2.7: Flow chart of Expectation-Maximization (EM) algorithm

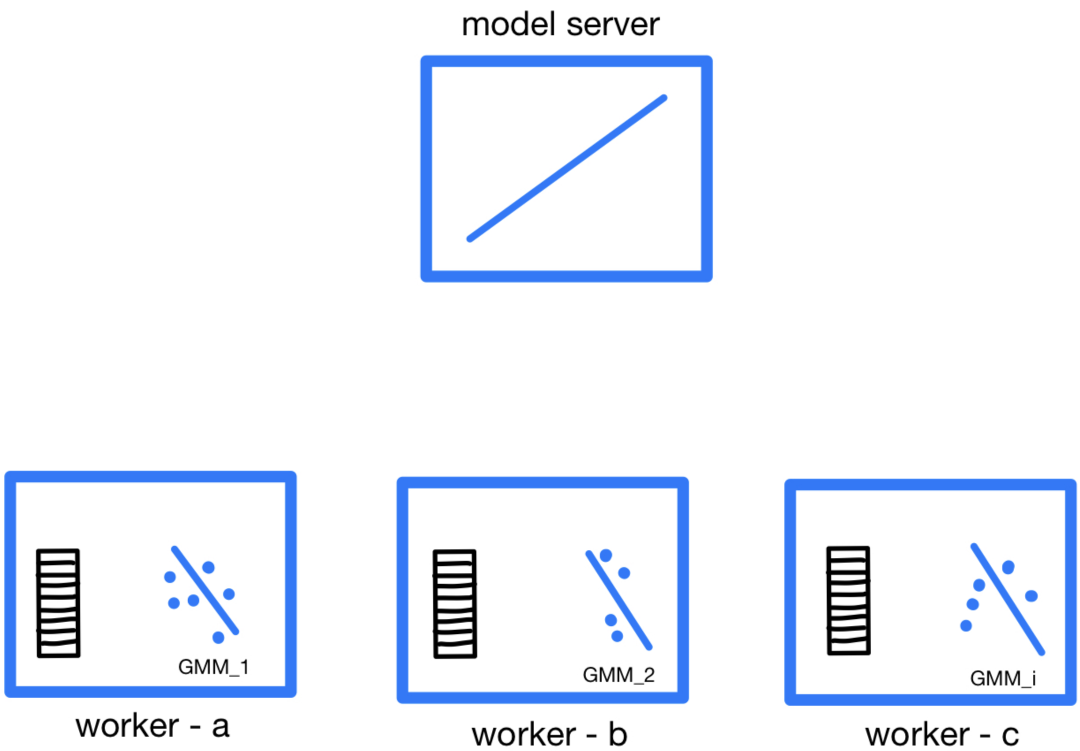

Figure 2.8: Local clients Building GMM on their data

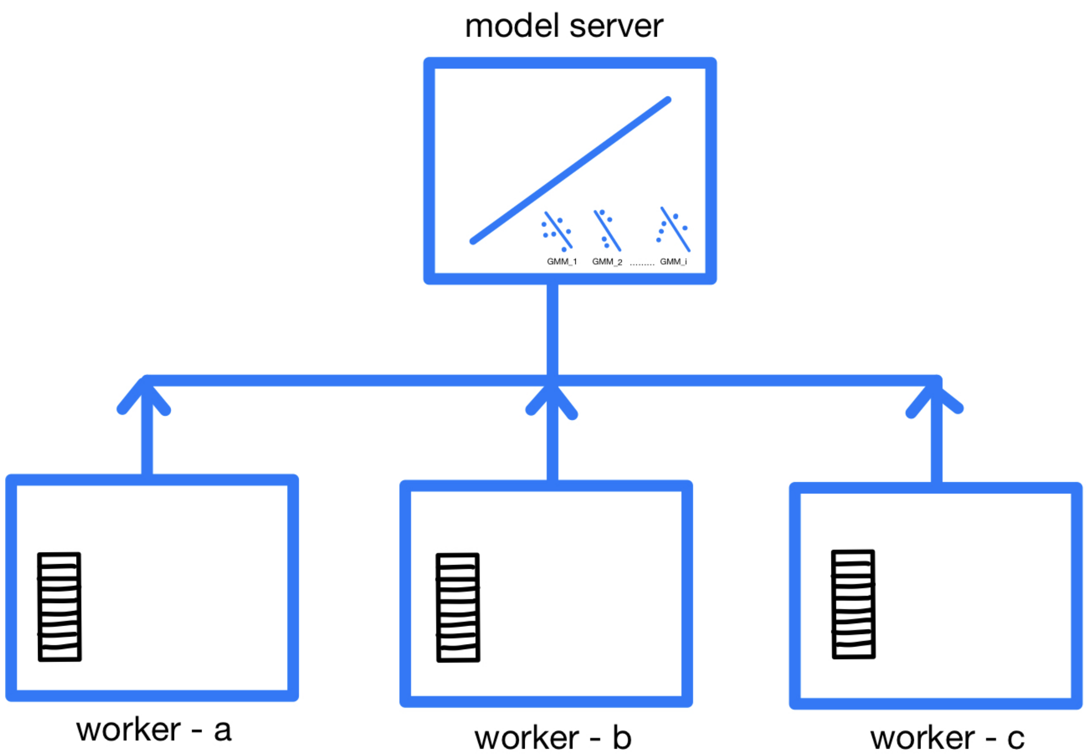

Figure 2.9: Clients sent GMMs to server

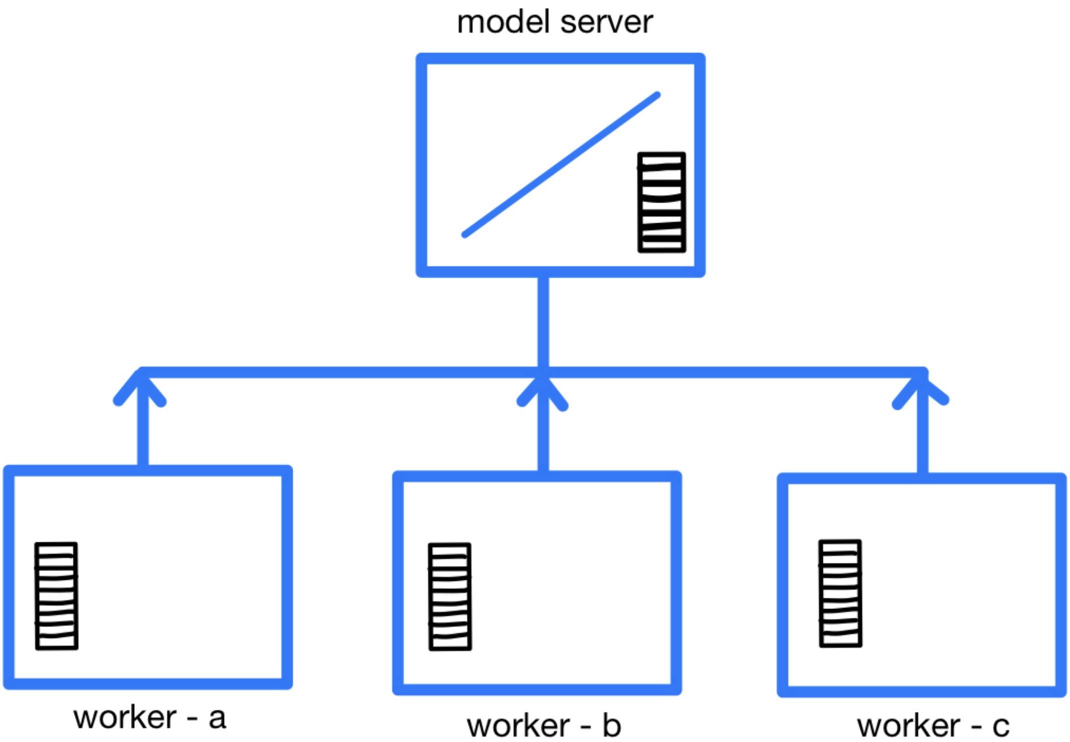

Figure 2.10: Data points sampled from clients' GMM

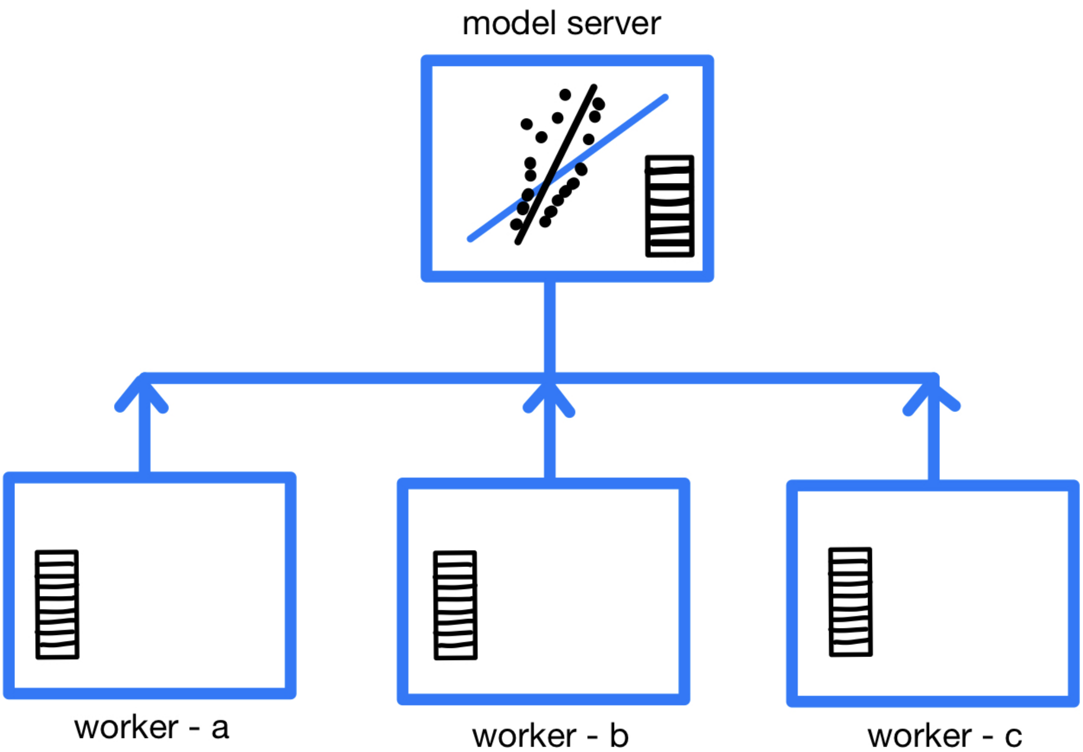

Figure 2.11: Central model trained on the sampled data

Pseudocode 3: Vanilla FL paradigm using GMM

```
Function OneshotLearning(N)
    N : number of clients
    Server :
    listening(gmm_list)
    D‘ = [] // initially empty
    for i∈ 1tolen(gmm_list) do
        D‘_ix <- sampling(gmm_i)
        D‘_iy <- label(i)
        D‘.append(D‘_i, D‘_iy)
    D‘_train , D‘_test <- train_test_spilt(D‘)
    model_c < −random_model()//any random model
    compile(model_c, loss_function, optimizer)
    model_c.fit(D‘_train)
    accuracy = model_c.evaluate(D‘_test)

    Client :
    D < −load_data()
    for i∈ 1toN do
        D_i <- distribute_data(D) // non-IID distribution
        // D_i is the private data of i^th client
    silhoutte_val = []
    for i∈ 1toN do
        sil_val_i <- silhoutte(D_i) // to find number of natural cluster
        silhoutte_val.append(sil_val_i)
    gmm_list : []
    for i∈ 1toN do
        n_comp = sil_val_i gmm_i <- build_gmm(D_i,n_comp) // non-IID
          distribution
        gmm_list.append(gmm_i)
    return gmm_list
```

CHAPTER 3

Literature Review

3.1 Communication-Efficient Learning of Deep Networks from Decentralized Data

In the article, H.Brendan McMahan et al. report, the FedAvg algorithm takes fewer number communication rounds to achieve the baseline accuracy than standard SGD. Experiments in this paper show that federated learning is practically possible as FedAVG algorithms train high-quality models in fewer communication rounds on various datasets. One communication round 3.1 in FedAVG is equivalent to one mini-batch update in standard SGD. There a slight variation of the algorithm where C = 1 (total client participation) over full dataset i.e. B = ∞ for single epoch (E = 1) discussed and named FedSGD. FedAVG gives more computational power to local clients by allowing cetrain number of epochs before gradient descent update.
They have done demonstrated extensively FedAVG result comparison with standard SGD and FedSGD over MNIST, CIFAR-10 and LTSM datasets using 2NN and deepCNN models. FedAVG givind (10-87)x speedup over standard results.

- Input : dataset , datset distribution, model , algorithm
- Output : interger x (number of rounds to achieve centrain accuracy)
- Process : model [A.1, A.2, A.3] over mentioned algorithm

Our approach is similar to the standard approach, but instead of going for multiple round, in one-shot using GMMs this can be done. We are building GMMs over local clients dataset. After that these GMMs are sent to central server and sampling is done. Since by building GMMs, local clients private data distribution is captured at server. And training done using these proxy data gives better results.

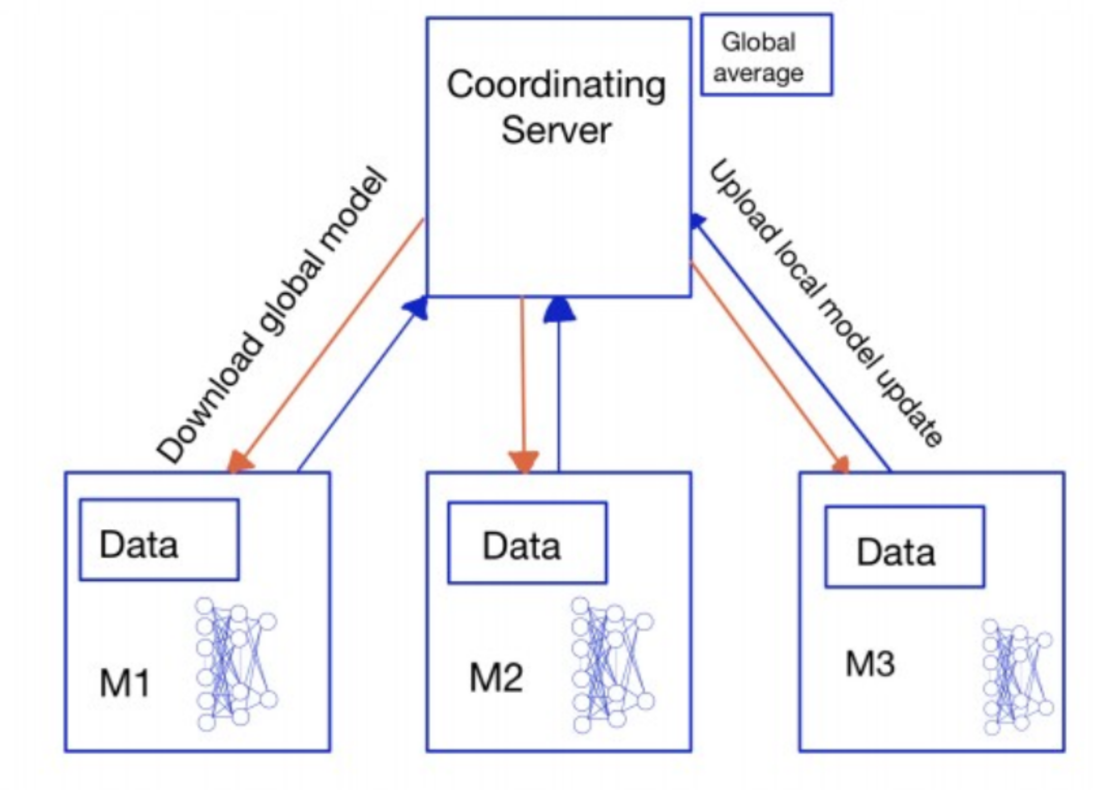

Figure 3.1: one communication round

3.2 Privacy-Preserving Asynchronous Vertical Federated Learning Algorithms for Multiparty Collaborative Learning

This article [5] talks about federated learning over vertically partitioned data. Vertically partitioned(VP) data are those data that belongs to a particular entity. But, these data are held by a different organization, and these data are mutually exclusive, and all will have different features about the same entity. Most of the available solution of federated learning on VP data is synchronous in nauter with is inefficient for VP data. So Bin Gn et al. proposes a three new asynchronous algorithm for federated learning on VP data while keeing the data privacy. These are AFSGD-VP, AFSVRG-VP, and AFSAGA-VP. Our approach is totally different than this algorithm. As our working algorithm is synchronous nature.

3.3 Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data

Felix Sattler et al. explains privacy preserving federated learning with less communication round comes at a cost of high communication overhead while training. The overhead can be equivalent eqaution (3.1) [9] to

$$b^{up/down} = N_{iteration} * f * |W| * (H * \Delta W^{up/down} + \eta) \tag{3.1}$$

where,

$N_{iteration}$ = total number of training iteration

f = communication frequency

|W| = size of the model

$\Delta W = W_{new} - W_{old}$

$H * \Delta W^{up/down}$ = entropy of the weight updates exchanges during the up/down

η = inefficiency of the encoding There are there possibilities to reduce the overhead keeping $N_{iteration}$ and |W| to be fixed.

- we can reduce the communication frequency f
- reduce $H * \Delta W^{up/down}$ via lossy compression
- use more efficient encoding to communicate the weight updates, thus reducing η.

The authors came up with a framework STC(sparse ternary compression) which can address the above three problems. Sparsification is a method to reduce entropy by limiting chances to small set of parameters. Only gradient with a magnitude greater than threshold are sent to server rest accumulated in the residual.This approach can be understood as an alternative paradigm for communication-efficient federated optimization that relies on high-frequent low-volume instead of low-frequent high-volume communication. It has shown through expreiments that it converges faster than federated averaging with respect to number of iteration.

CHAPTER 4

Procedure, Techniques and Methodologies

4.1 Datasets

A good image classification model in mobile phones and real-time devices can enhance the user experience. So we are motivated to first test the approach to image data. For this task, we have chosen a proxy dataset of modest size to tune the hyperparameter of our algorithm. Our initial experiments ran on the MNIST dataset, and later extended_MNIST and CIFAR-10.

As federated optimization draws its connection from distributed optimization, data distribution among clients is typical federated optimization that can significantly impact model learning. In this regard, our datasets are distributed among clients in two manners, IID, and Non-IID

- **IID (Independent and Identical distribution)** in this whole dataset is shuffled randomly and then equally distributed among clients. For example, we have 100 clients and for CIFAR-10 datasets which is having 60000 data points. Each client will receive 600 data points randomly.
- **Non-IID** in this we first filter the data points specific to labels and then shuffled. It then distributed among clients in label specific manner. For example, we have 100 clients and for CIFAR10 datasets which is having 60000 data points. We filter the data points specific to labels. 6000 for each labels. Each clients will receive 600 data points specific to one label.

4.1.1 MNIST

Each datapoint is a 8$*$8 image of integer, each pixel range $\in$ [0,255]. This is a toy dataset 5.2 in sklearn library.

4.1.2 extended_MNIST

Each datapoint is a 28*28 image of integer, each pixel range $\in$ [0,255]. This dataset 4.2 is good for trying pattern recognition methods and learning problems. It is based on

Table 4.1: MNIST dataset details

Key	Value
# of classes	10
# of features	16
Dimensionality	64
Total samples	1794
Sample per class	approx 180

real-world data and requires less effort in cleaning and formatting.

Table 4.2: extended_MNIST dataset details

Key	Value
# of classes	10
# of features	16
Dimensionality	784
Total training samples	60000
Total testing samples	10000
Samples per class	6000

4.1.3 CIFAR-10

That CIFAR-10 dataset consists of 32*32*3 color images. This dataset 4.3 is good for trying pattern recognition methods and learning problems. It is based on real-world data and requires less effort in cleaning and formatting. There are 10 different types of image classes namely ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck'], each classes are mutually exclusive. For example, car and truck class intersection have zero images.

Table 4.3: CIFAR-10 dataset details

Key	Value
# of classes	10
Dimensionality	3072
Total training samples	50000
Total testing samples	10000
Samples per class	5000

4.2 Model

4.2.1 For MNIST dataset

We have used random forest a supervised machine learning model which is very popular and widely used in classification and regression task.

4.2.2 For extended_MNIST dataset

For this we have worked on two neural network models. First, a sequential model with 2 fully connected hidden dense layer, 200 units each with relu activations(199,210 total parameters) A.1 [7]. Second, a CNN model with two 5X5 convolution layers each followed by (2X2) max-pooling layers followed by dense layer, 512 units each with relu activations then softmax layer. A.2 [7]

4.2.3 For CIFAR-10 dataset

A deep CNN model with three 3X3 convolution layers, first two followed by (2X2) max-pooling layers followed by dense layer, 128 units each with relu activations then softmax layer. A.3 [11]

4.3 Expermients

4.3.1 Experiment 1 : Sanity test of GMMs as proxy data in vanilla federated learning settings

Aim

To study the effect on the performance of a central model on adding synthetic data sampled from local GMMs.

Expected outcome

Accuracy of the centralized model formed by adding synthetic data should be more than the model created in absence of the synthetic data. The local models should get enriched.

Observed outcome

Accuracy of the centralized model on local data sets after adding synthetic data is more than its accuracy when the model was not trained on synthetic data points. The updated centralised model have also lead to an improvement of the local models.

Methodology

Pseudocode 4: Central model update using synthetic data points

```
Function ()
    Sample D <- load MNIST digits dataset
    Label_index = []
    for i ∈ [0,1,....,9] do
        for i ∈ length(D) do
            extract label specific index & append to Label_index

    Central dataset: C_data < −ϕ
    for i ∈ [0,1,....,9] do
        C_data <- C_data ∪ extract (first 5 data points with label i using Label_index)
    C_model : initialise(RandomForest model)
    C_model <- train(C_data)

    for i ∈ [0,1,....,9] do
        create LocalDataset_i
        LocalModel_i : initialise(RandomForest model)
        LocalModel_i <- Train( LocalDataset_i)
        Accuracy_i <- Score( C_model , LocalDataset_i )
    // Creating GMMs Sending to central server
    for i ∈ [0,1,....,9] do
        GMM_i <- CreateGMM(LocalDataset_i)
    // Sampling synthetic data
    SyntheticDataset <- ϕ for i ∈ [0,1,....,9] do
        SyntheticDataset <- SyntheticDataset ∪sampleData(GMM_i)
    C_data <- C_data ∪ SyntheticDataset
    C_model <- Train( C_data ) for i ∈ [0,1,....,9] do

        Accuracy_i <- Score( C_model , LocalDataset_i )
```

Inferences

Accuracy of the central model increases this implies that our GMMs is working good as proxy data of our local clients. But, while creating the GMMs for our local clients, number of components are fixed which is hyperparameter and essential for the learning of GMMs. So, further we are looking for finding the optimal number of clusters for each of the dataset using silhouette.

Result

Table 4.4: Accuracy of Central Model on updating using GMMs(synthetic data)

Digit Set (Local Clients)	Accuracy Before GMMs	Accuracy After GMMs
0	98	100
1	84	84
2	95	95
3	92	96
4	92	92
5	76	91
6	96	97
7	94	98
8	70	84
9	84	92

Conclusion

We see an increase in the central model accuracy for local clients. Since its a toy dataset its not concrete. So, to further prove its concrete evidence we demonstrate on real world datasets and compare results with federated averaging.

4.3.2 Experiment 2 : Sanity test of weight average mechanism

Aim

To observe the impact of weight update of central model based on averaging of NN model A.1 of local clients and to verify if passing the parameters can lead to a better accuracy for the central model

Expected outcome

Accuracy of the centralized model after weight update will increase.

Observed outcome

The performance of the NN model A.1 is low. As neural networks are non-linear models there is no guarantee if the simple averaging will improve the performance of the central model or not.

Methodology

Please refer Chapter 2 Pseudocode 1.

Result

Accuracy of neural network before averaging is 85%.

Accuracy of neural network after averaging is 64%.

Inferences

NN model is a non-linear model, simple mean averaging may or may not work.

Conclusion

There is no clear understanding of why simple averaging fails while weights update on NN model. Our assumption is not true always.

4.3.3 Experiment 3 : Simple comparison of weight average vs GMM approach

Aim

To compare weighted average approach against GMM approach for central model performance

Expected outcome

GMM approach should give better performance than weighted average.

Observed outcome

GMM approach has given 10% better accuracy than weighted averaging of NN.

Methodology

Result

Mean accuracy after update using GMM is 78%.

Mean accuracy after update using weighted average is 69%.

Conclusion

Instead of sending gigabytes of data as parameters to central server, which itself is a problem [9] , we can send GMMs and achieve better results.

Pseudocode 5: weighted average and proxy data for model update

```
Function ()
    Sample D <- load dataset
    Central dataset: C_data <- φ
    for i ∈ [0,1,....,9] do
        C_data <- C_data ∪ extract (30% data points with label i using Label_index)
    C_train , C_test <- Split( C_data )
    C_model <- Train( C_train)
    OriginalAccuracy <- Score( C_model, C_test )
    for i ∈ [0,1,....,9] do
        Model_i <- C_model // Creating 10 copies of C_model
    WeightSum <- Init(shape( C_model ), 0 ) // assign shape of Cmodel with all 0's
    for i ∈ [0,1,....,9] do
        Create LocalDataset_i
        GMM_i <- CreateGMM( LocalDataset_i )
        Train ( Model_i , LocalDataset_i )
        WeightSum <- WeightSum + getWeight( LocalModel_i )
    // Synthetic data generation
    SyntheticDataset <- φfor i ∈ [0,1,....,9] do
        SyntheticDataset <- SyntheticDataset ∪ sample(GMM_i)
    C'_model <- C_model
    AverageWeights <- WeightSum / Number of Model_i
    C_model <- ReconfigureWeights ( C_model , AverageWeights )
    AveragedAccuracy1 <- Score( C_model, C_test )
    C_data <- C_data ∪ SyntheticDataset
    C'_model <- Train( C_data )
    AveragedAccuracy2 <- Score( C'_model, C_test )
    compare(AveragedAccuracy1, AveragedAccuracy2)
```

CHAPTER 5

Results and Discussions

With our extensive experiment on different datasets and on different models with different data distribution we observed great results.

For IID data distribution

Table 5.1: IID data distribution results

Dataset	Model Name	Accuracy
MNIST	2NN Model	95 (in 85 rounds)
MNIST	CNN Model	95 (in 18 rounds)
CIFAR-10	CIFAR-10 Model	80 (in 380 rounds)
MNIST	One-shot using GMM	-
CIFAR-10	One-shot using GMM	-

For Non-IID data distribution

Table 5.2: Non-IID data distribution results

Dataset	Model Name	Accuracy
MNIST	2NN Model	87 (in 420 rounds)
MNIST	CNN Model	82 (in 131 rounds)
CIFAR-10	CIFAR-10 Model	70 (in 3100 rounds)
MNIST	One-shot using GMM	42
CIFAR-10	One-shot using GMM	30

CHAPTER 6

6.1 CONCLUSION

This work shows from our extensive experiments that one-shot learning using GMM probabilistic model is feasible. Since it is possible to train high-quality model using this approach. As demonstrated by results in our experiments on MNIST and CIFAR10 datasets, it indeed clear that this is working on spatial data or image data, but it can be apply to other datasets also with careful observation. Our approach is also offering privacy of client's data using proxy-data.

6.2 SUMMARY

Federated learning is a new technique of machine learning in which a model is trained by combining the updates received from clients without revealing the client's data. There are two approaches to central server update:

- By sampling synthetic data using GMMs
- By sending weights to the central and aggregation

This reports discuss approaches like FedSGD, FedAvg and One-shot learning using GMMs in great details. FedAvg is the more powerful version of FedSGD as it offer more computation to local clients by allowing more of gradient updates. On the other hand One-shot learning using GMMs is a different approach where instead of averaging or doing any statistical averaging we consider a thought of proxy data for actual data.

REFERENCES

[1] (2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance). URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679.

[2] **M. ANDERSON** (2015). Technology device ownership: 2015. URL https://www.pewresearch.org/internet/2015/10/29/technology-device-ownership-2015/.

[3] **A. Anonymous** (2013). Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *Journal of Privacy and Confidentiality*, **4**(2). URL https://journalprivacyconfidentiality.org/index.php/jpc/article/view/623.

[4] **D. R. Brendan McMahan** (2017). Federated learning: Collaborative machine learning without centralized training data. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[5] **B. Gu**, **A. Xu**, **Z. Huo**, **C. Deng**, and **H. Huang** (2021). Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning. *IEEE Transactions on Neural Networks and Learning Systems*, 1–13.

[6] **C. Guo**, **H. Fu**, and **W. Luk**, A fully-pipelined expectation-maximization engine for gaussian mixture models. *In 2012 International Conference on Field-Programmable Technology*. IEEE, 2012.

[7] **H. B. McMahan**, **E. Moore**, **D. Ramage**, **S. Hampson**, and **B. A. y. Arcas** (2016). Communication-efficient learning of deep networks from decentralized data. URL https://arxiv.org/abs/1602.05629.

[8] **J. POUSHTER** (2016). Smartphone ownership and internet usage continues to climb in emerging economies. URL https://www.pewresearch.org/global/2016/02/22/smartphone-ownership-and-internet-usage-continues-to/.

[9] **F. Sattler**, **S. Wiedemann**, **K.-R. Müller**, and **W. Samek** (2020). Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, **31**(9), 3400–3413.

[10] **Sklearn GMM Documentation** (2017). Two-component gaussian mixture model.

[11] **TensorFlow team** (2022). Tensorflow convolutional neural network (cnn) tutorial, 2022. URL https://www.tensorflow.org/tutorials/images/cnn.

[12] **Q. Yang**, **Y. Liu**, **T. Chen**, and **Y. Tong** (2019). Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, **10**(2). ISSN 2157-6904. URL https://doi.org/10.1145/3298981.

APPENDIX A

Supplemental Figures and Tables

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 200)               157000
_________________________________________________________________
dense_25 (Dense)             (None, 200)               40200
_________________________________________________________________
dense_26 (Dense)             (None, 10)                2010
=================================================================
Total params: 199,210
Trainable params: 199,210
Non-trainable params: 0
_________________________________________________________________
```

Figure A.1: 2NN model summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_34 (Dense)             (None, 512)               1606144
_________________________________________________________________
dense_35 (Dense)             (None, 10)                5130
=================================================================
Total params: 1,663,370
Trainable params: 1,663,370
Non-trainable params: 0
_________________________________________________________________
```

Figure A.2: MNIST CNN model summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

Figure A.3: CIFAR-10 model summary