# Federated Distribution Learning

*A Project Report*

*submitted by*

**MEHAK BANSAL**

*in partial fulfilment of the requirements*
*for the award of the degree of*

**MASTER OF TECHNOLOGY**

COMPUTER SCIENCE AND ENGINEERING
INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

**Jan, 2024**

# THESIS CERTIFICATE

This is to certify that the thesis titled **"Federated Distribution Learning"**, submitted by **Mehak Bansal**, to the Indraprastha Institute of Information Technology, Delhi, for the award of the degree of **Master of Technology**, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree. The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

**Dr. Bapi Chatterjee**
Thesis Supervisor
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology Delhi
New Delhi 110020

Date: 19th January 2024

# ACKNOWLEDGEMENTS

I am immensely grateful to Professor Dr. Bapi Chatterjee for his confidence in me and for granting me the opportunity to work under him on my thesis. I would like to express my sincere gratitude for his invaluable guidance and unwavering support throughout this research work. I am truly honored by his trust in me. He has been a constant source of inspiration and motivation for me.

I thank my family and friends for their unconditional love and encouragement. They have always stood by me in times of difficulty and joy. I want to thank my labmates, especially **Drishya Uniyal** for their help and cooperation, who have shared their insights and expertise with me. They have always provided constructive feedback and suggestions for improving my work. I am also indebted to the IIITD administration and IIITD Computer Science and Engineering department for providing me with the necessary facilities, infrastructure and resources for conducting this research. I am grateful to all the people who have directly or indirectly contributed to this thesis. I appreciate their time and effort.

# ABSTRACT

Distribution Learning is a form of machine learning which involves the distribution of data across in a network having multiple devices. Each node is trained independently, and then combined the results to generate a model which is global model. This greatly accelerates and parallelizes the training process and optimizing time whereas, federated learning is a secure distribution machine learning paradigm which is specifically designed for privacy-sensitive scenarios where the data stays on local devices of users, and models are trained on the their devices itself without the sharing of the data with the central server. To enhance the capabilities of Federated Learning, Mixture Density Networks are integrated into the federated learning environment. MDNs are neural networks that can simulate complex probability distributions and measure prediction uncertainty. In this study, MDN is implemented in three distinct Federated Learning settings and compares their performance. The primary objectives of this research are to investigate the use of MDNs in distribution learning and evaluate the effectiveness of federated learning. The experiment focuses specifically on exploring the federated learning approach for distribution learning, with a direct emphasis on the distribution itself rather than other properties that may affect the data.

KEYWORDS: Federated Learning ; Distribution Learning ; Mixture Density Networks ; Privacy Protection

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **IID** | Independent and Identically Distributed |
| **FL** | Federated Learning |
| **MDN** | Mixture Density Network |
| **MSE** | Mean Squared Error |
| **GMM** | Gaussian Mixture Models |

# CHAPTER 1

# INTRODUCTION

**Google** introduced the groundbreaking concept of Federated Learning . Federated Learning has the ability to offer model training in a distribution manner, allowing for learning from immense data collection. This makes it the preferred choice for organizations seeking to leverage the power of machine learning while maintaining the confidentiality of device data as the data itself is not being shared. Unlike traditional approaches, Federated Learning eliminates the need to centrally collect raw data from each participating device, ensuring data privacy and security.

Distribution learning, on the other hand, is a machine learning technique where data is distributed across multiple devices or nodes within a network; each node is independently trained, and the results are combined to generate a final model. This approach accelerates and parallelizes training on abundant data, optimizing time and resources. The primary objective of Distribution Learning is to harness the collective computational power of these distributied devices to expedite the training process, enhance scalability, and effectively handle complex models or larger datasets through parallel processing. This approach efficiently manages the computational load. Federated Learning takes distribution learning to the next level by providing a secure paradigm for privacy-sensitive scenarios. In FL, data remains on local devices, and models are trained directly on users' devices without sharing the data with a central server. The updated weights of the trained models from each node are then sent to a central server, where they are aggregated to improve the global model. This process ensures that sensitive data remains on users' devices, enhancing privacy and data protection.

By harnessing the power of Federated Learning, organizations can exploit distributed data to generate high-performance models capable of producing intelligent responses. This opens new possibilities for machine learning applications along with the addressing of security and privacy concerns. It eliminates the need for exchanging the raw data between client devices and global servers. Instead, the raw data residing on edge devices is utilized to locally train the model, thereby keeping data privacy. The organizations face increasing challenges in collecting and sharing data. Kinds of financial

data or medical records, are of a sensitive nature and cannot be freely shared. Thus, such data sets are confined within isolated data silos, managed solely by the data owners; which is obstacle to user privacy, data security, and even data integration within the same organization. Data silo is a collection of data possessed by one group within an organization, which makes difficult for other groups to access or utilize that data effectively. This lack of accessibility arise many challenges, hindering collaboration and overall productivity. Thus, data silos create barriers between teams, increase costs, result in poor customer experiences and limit adaptability. Combining data dispersed across multiple institutions becomes nearly impossible due to the exorbitant costs involved. Consider a scenario where two organizations aim to collaborate on constructing a machine learning model using medical data. When data is transferred using traditional approaches from one organization to another, the original data owner often loses control over their own data. The value of the data diminishes the moment it leaves their possession.

## 1.1 Background

**Satellites** capturing images of the Earth faces challenge in sending the data they collected to the data centers at ground. The sheer volume of information makes it impractical to send everything. Likewise, autonomous vehicles must seamlessly interact with other cars and computing centers worldwide, while also processing a significant amount of data locally using machine learning models. The existing computing methods struggle to address the crucial issue of securely and efficiently sharing and updating these models across different locations. Efficient transmission of data collected by satellites faces obstacles. The information captured by these satellites and to transmit every single piece of data to the data centers at ground is difficult. Similarly, the success of **autonomous vehicles** hinges on their ability to communicate and collaborate with other vehicles and computing centers across the globe. These vehicles must process the data locally, utilizing the machine learning models. However, the current computing strategies fall short in addressing the critical challenge of securely and efficiently sharing and updating these models across diverse locations.

The transmission of data collected by Earth-observing satellites and the seamless

collaboration of autonomous vehicles with other cars and computing centers present significant challenges. The existing computing methods must grapple with the complex task of securely and effectively sharing and updating machine learning models across diverse locations. Addressing these obstacles requires a professional and innovative approach that prioritizes data integrity, accessibility, and efficiency. The existing computing approaches face issue of devising a secure and effective method to facilitate the sharing and updating of machine learning models across multiple sites. This demands a solution that not only ensures the confidentiality and integrity of the models but also optimizes their accessibility and efficiency.

**IID Data** is Independent and Identically distributed data. Each data point is independent of all other data points. Thus, the outcome of one data point does not affect other data points' values or outcome. All the data points have same probability distribution, i.e., the statistical properties of the data, such as mean, variance, and probability distribution, are same for every data point. Each data point is a random sample from the same distribution. In real-world scenarios, we cannot always assume that data is IID. Dependencies between data points, temporal patterns, or other factors can violate the independence assumption.

**Non-IID data** is non-independent and identically distributed data. The data points are not statistically independent. Thus, the outcome of one data point may be influenced by or correlated with the values or outcomes of other data points. Dependencies or correlations can exist between data points. The data points are not from the same probability distribution. Different subsets of data may follow different distributions or their statistical properties may possess variations. Thus, the statistical properties, like mean, variance, or probability distribution, are not the same for all data points.

**Time Series Data:** Data collected over time possess temporal dependencies, making them non-IID. For example, stock prices, weather measurements, or sensor data collected at different time points may show correlations.

**Grouped or Clustered Data:** Data grouped by categories or clusters may exhibit non-IID characteristics. For instance, customer data can vary significantly between different customer segments.

**Data from Heterogeneous Sources:** Data collected from different sources or sen-

sors can have different distributions or properties, making them non-IID.

## 1.2 Problem

Organizations encounter numerous challenges when it comes to gathering and sharing sensitive data, resulting in the fragmentation of valuable information. Traditional methods of data sharing often pose risks to privacy and data security. However, a solution named Federated Learning has emerged to address these issues by keeping data on local devices. Despite this advancement, certain challenges persist in terms of ensuring transparent data integration and equitable distribution of benefits. Furthermore, the research acknowledges the hurdles associated with efficiently sharing and updating machine learning models across different locations, particularly in scenarios involving Earth-observing satellites and autonomous vehicles. These complex environments demand seamless collaboration and coordination. Moreover, the concept of IID data is introduced, emphasizing the necessity of addressing dependencies and correlations in non-IID data. In real-world scenarios such as time series data, grouped or clustered data, and data from heterogeneous sources, it is common to encounter non-IID data. Therefore, it becomes crucial to develop strategies that account for these variations and complexities. By recognizing these challenges and exploring innovative solutions, organizations can overcome the obstacles hindering effective data sharing and integration. This will ultimately lead to enhanced collaboration, improved data security, and more equitable distribution of benefits.

## 1.3 Purpose

The research aims to explore how Mixture Density Networks, a type of neural network (combination of neural network and gaussian mixture model) known for handling complex probability distributions, can be used in a federated learning setup. MDNs are good at dealing with uncertainty in predictions. The goal is to understand the challenges of training MDNs, especially in scenarios where data is spread out in different locations, and to see if federated learning can help overcome the issues. The study wants to add to our knowledge about federated learning and MDNs. It aims to show that combining

these approaches can create high-performance models while also addressing privacy concerns. Specifically, the research focuses on using Federated Learning to deal with distribution learning, emphasizing the distribution of data rather than other factors that might affect the data.

## 1.4 Outline

The rest of the thesis is organized as follows:

- Chapter 2 contains the literature review of few research papers of federated learning
- In Chapter 3, there is the introduction to FL and its aggregation algorithms.
- In chapter 4, MDN is explained in detail.
- Chapter 5 contains the training process.
- In Chapter 6, There are the results from the experiments and conclusion.
- Chapter 7 contains the proposed potential future works.

# CHAPTER 2

# LITERATURE REVIEW

The paper presented by Li et al.(10) is a preliminary work in Federated Learning(FL). The authors first explain the need for federated learning to aid distribution learning approaches. They describe how Federated Learning can help preserve the data owners' privacy while simultaneously using that data's vastness to train high-performance models like deep CNNs. The paper mentions IoT, healthcare, and smartphones as some significant application areas of Federated Learning. The paper then presents the core challenges of FL to the reader. The paper finishes by highlighting extreme communication schemes, communication reduction, the Pareto frontier, novel models of asynchrony, heterogeneity diagnostics, and granularity privacy constraints as a few future directions.

Liu et al.(5) comprehensively provide a survey of the work done on federated learning. The paper proposes a four-layer architecture of federated learning and taxonomy of the techniques related to FL. The authors first give an overview of federated learning followed by an explanation of the few aspects of federated learning systems, including various parallelism types, aggregation algorithms, data manipulation techniques, Etc. The paper explains how federated learning systems efficiently exploit distribution computing resources such that the machine learning models can be collaboratively trained and the raw data need not be shared to the global server, thus providing data security and privacy protection to the end user as well as by using the encryption technique for network connections. The paper presents various widely used open-source frameworks of FL based on their functional architecture, used in domains like NLP, next-word prediction in Android, and many more, which mainly use centralized aggregation algorithms. The paper concludes with the limitations and new research directions of the frameworks of federated learning systems.

Rodolfo et al.(4) present a systematic literature review on the current research on FL in the context of Electronics Health Records data for healthcare applications. To do this, they carefully select a literature corpus of 67 articles. It turns out from the analysis that

30% of these articles are about the case studies of FL on medical data, 28% are about confidentiality guarantees for training data, 18% discuss aggregation methodologies for global model generation, 16% discuss architectures of FL in medical applications, and 7% are on Federated Data Analysis. These articles demonstrate the methods to propose and improve methods to enhance the confidentiality of training data. Many studies explore the application of FL on various types of medical data. Model aggregation methods and FL architectures are specific to the context of medical data are two recent domains where research is currently on a full scale.

Sheller et al.(27) illustrate the efficiency of using Federated Learning in the medical context. The authors experiment by using collaborative learning among ten participating institutions. This experiment ensures a data private multi-institutional collaboration to learn a model without compromising the data confidentiality. The quality of the model thus obtained was found to be 99 percent of the quality achieved when the same model was trained in a centralized learning framework. The paper states that although such a collaborative approach introduces some errors, its advantages far outweigh those errors.

Ng et al.(13) present the application of Federated Learning to aid radiologists in performing better and faster diagnoses based on medical imaging like CT scans, X-Ray, sonography, and MRI scan. The paper mentions cases where deep CNN models have outperformed radiologists in making these diagnoses. However, to tap the potential of these models, we need vast and diverse datasets, which is a distant possibility in the medical context due to several ethical and legal codes. Federated Learning is an alternative in such scenarios as it alleviates the need for the data to be shared centrally. The paper's authors discuss two experiments conducted using the publicly available dataset from the brain tumor segmentation challenge 2018. The first experiment, led by Intel, measured the performance of a federated semantic segmentation model and found that the results obtained were comparable to those obtained when the model was trained using a central dataset. The second experiment, led by Nvidia, applied a differential privacy technique in the Federated Learning setting to safeguard the patient data better while ensuring not to hamper the model performance. It found that the segmentation results thus obtained were comparable to those obtained without applying differential privacy.

Ammar et al.(15) present the use of Federated Learning for Covid 19 screening from chest X-ray images. Healthcare institutions can primarily benefit from using Federated Learning as they can tap the power of deep learning models that require extensive and diverse datasets without sharing their data centrally. The authors compare classical centralized and federated learning using two deep CNN models, VGG16 and ResNet50, in both frameworks. The results obtained for both learning approaches were comparable and even equivalent after a few epochs. Thus, the paper establishes the effectiveness of Federated Learning in diagnosing Covid 19 using chest X-rays while also alleviating privacy concerns over sharing of medical data.

Hao et al.(16) present a privacy-enhanced Federated Learning ( PEFL ) scheme. The attackers can cause much harm to many Industrial applications by exploiting the parameters shared in a Federated Learning setting, even though the raw data is not shared. PEFL prevents the leakage of information from the shared parameters as well as the local gradients. It provides high privacy protection even when an adversary colludes with multiple honest entities. PEFL guarantees security oblivious of aggregation. It can do so as it is non-interactive in each aggregation at the central server. The authors also evaluate if the use of PEFL affects the performance of a Federated Learning model in a negative direction. To test this, they measure the performance of PEFL on the MNIST dataset and observe that the model achieves practical accuracy.

Kang et al.(24) address the vulnerability in Federated Learning settings introduced by the dependence of the central server on the devices. The devices may, intentionally or unintentionally, mislead the global model by sending malicious updates. The authors present the mechanism of associating a reputation measurement for each device to filter in the parameters only from the reliable devices based on their reputation. The authors use a multi-weight subjective logic model to compute the reputation of each device. In order to manage the reputation of devices in a decentralized manner, the paper discusses the use of a consortium blockchain. Towards the end, the authors demonstrate that this additional work of filtering out unreliable devices adds security and provides the performance equivalent to the case when this filtering is not employed.

Luo et al.(6) present a cost-effective design for Federated Learning. The paper discusses a multivariate control problem to implement FL in a cost-efficient manner while also guaranteeing convergence. The authors choose two control variables, the number

of participating clients and the number of local iterations in each FL round, to minimize the implementation cost regarding learning time and energy consumption. The paper also presents insights into solution properties that can help identify design principles for different optimization goals. The authors empirically establish the algorithm's robustness chosen to solve this optimization problem.

C. Ma et al.(7) address the issue that the single central server of the federated learning framework can behave maliciously. The authors proposed a blockchain-assisted federated learning architecture called BLADE-FL. They investigated this framework which is decentralized in nature such that it prevents the poisoning of the training process by malicious clients. The paper provides a federated learning framework free from malicious users to provide a reliable environment for other clients. This framework uses a decentralized aggregation algorithm, and the training of federated learning systems and blockchain mining is integrated. The paper concludes with the issue found in users' privacy in the proposed framework that the models shared by the other users can be duplicated by a lazy client so that the lazy client can get advantages without having his computational resources used in the federated learning. The paper proposes a few possible solutions to address this issue: noise can be added, and lazy clients can be detected by achieving differential privacy locally and using PN (pseudo-noise). The paper concludes with the future directions to study in the area related to FL.

# CHAPTER 3

# FEDERATED LEARNING

## 3.1 Introduction

Federated Learning is a distribution machine learning paradigm specifically for privacy-sensitive scenarios. It allows data to remain securely stored on local devices, avoiding the need to share sensitive information with a central server. Models are trained directly on users' devices, ensuring utmost privacy and confidentiality. The updated weights of these trained models are then transmitted to a central server, where they are aggregated to enhance the global model. By leveraging distributed data, FL systems generate high-performance models that yield intelligent responses.

The architecture of Federated Learning has multiple data owners, a robust communication infrastructure, and a federated learning manager. The data owners, which are individual devices, collaborate seamlessly to perform the learning task. Federated learning manager acts as the central element of the federated learning framework. It manages the entire process and can be implemented using centralized approaches, such as cloud instances, or decentralized approaches, such as blockchain frameworks. Additionally, a reliable communication infrastructure is important to connect the devices with the central manager, ensuring smooth and efficient collaboration.

The way FL works is as follows:

1. The FL manager sends a copy of the global model to each of the data owners.
2. The data owners learn the model parameters using their local data and send these updated parameters to the FL manager.
3. The FL manager employs an aggregation methodology to combine the parameters received from each data owner in order to generate a global learning model.
4. Go back to step 1 until a satisfactory performance is achieved.

The fundamental concept behind FL is to train a model at each site where a data source is located and subsequently enable these locations to share their respective models until a global model is obtained. The communication method guarantees that no

site can guess or conclude some personal information about any other site, thereby safeguarding user privacy and data confidentiality. Simultaneously, the global model is constructed as if the data sources were combined, ensuring optimal performance and accuracy.

In summary, Federated Learning prioritizes privacy and security in the field of machine learning. By allowing data to remain on local devices and training models directly on users' devices, FL ensures that sensitive information is never compromised. With its robust architecture and carefully designed communication infrastructure, FL enables seamless collaboration among data owners, resulting in high-performance models that provide intelligent responses. This paradigm sets a new standard for privacy-sensitive scenarios, where user privacy and data confidentiality are of utmost importance. Federated Learning operates through a systematic approach that ensures efficient collaboration between the FL manager and data owners. By distributing the global model and allowing data owners to update parameters based on their local data, FL harnesses the collective intelligence of diverse datasets. The FL manager then skillfully combines these parameters to generate a powerful global learning model. This iterative process continues until the desired level of performance is attained. Through this collaborative and iterative approach, FL enables the development of robust and accurate models while preserving data privacy and security.

## 3.2 Need of FL

The emerging of mobiles led to the widespread adoption of federated learning, where each device learns a collaborative model while ensuring the privacy of its data stored locally. Unlike traditional distribution learning or centralized approaches, federated learning does not require devices to share data with each other. Instead, data is initially uploaded to a centralized server in the traditional approach, and parallelization is achieved by dividing this data among multiple clients for simultaneous training. This type of training aims to leverage the computing power in parallel. Even in classical decentralized learning, the data is assumed to be identically distributed, following an IID distribution. However, in federated learning, the data is known to be heterogeneously distributed across multiple devices in a non-IID setting.

Federated Learning alleviates the need for the data to be present centrally to apply Machine Learning and Deep Learning models. The data owners can collaborate to locally learn the parameters of the global model shared with all the data owners. The absence of the requirement for all the data to be present centrally ensures the privacy of the raw data. Although this form of collaborative learning paves the way to apply complex models, such as several deep CNN models that are otherwise difficult to apply due to the shortage of diverse and vast datasets, this type of learning has its challenges. These challenges include expensive communication, systems heterogeneity, statistical heterogeneity, and privacy concerns (although due to shared parameters). There are various vital aspects of Federated Learning. These are diverse types of aggregation algorithms, data parallelism, data communication, and the security of federated learning systems.

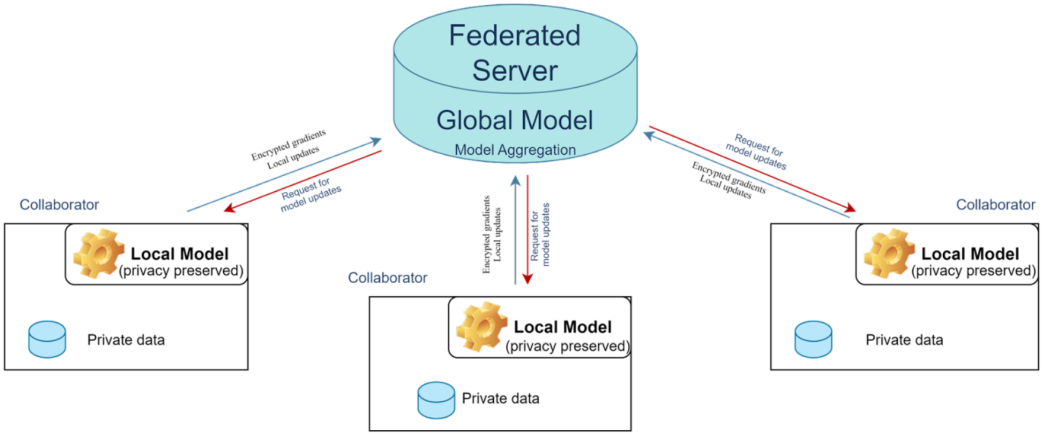


Figure 3.1: Federated Learning

Healthcare and the Internet of Things are two major application areas of Federated Learning. The healthcare sector can benefit mainly from FL as multiple studies have shown that many models, such as deep CNN, give outstanding results, sometimes even outperforming the experts in making diagnoses. However, to give reliable results, these models require large and diverse datasets at their base. Due to various legal and ethical codes binding healthcare institutions, this requirement is only sometimes satisfied in the case of medical data. Federated Learning comes as a solution in this scenario as it enables the system to tap into the potential of this vastness of data while preserving data confidentiality. Multiple hospitals can act as data owners in such an FL setting. A model thus built will be more reliable than one built using the centrally available data of

a single institution. It will be better trained to handle cases that might appear atypical to an institution due to factors like demography and ethnicity. FL can prove to be a boon to the medical field and can help reduce the workload on doctors and workers in this field. The following primary application is that of the Internet of Things( IoT). Modern IoT devices collect, adapt and react to the continuous stream of incoming data in real-time. Each device in such a network has limited connectivity, and the data is private. This makes building aggregate models in the domain of IoT difficult. Here again, FL offers a solution due to its privacy-preserving property.

Some of the improvements are implemented on the plain FL. Privacy Enhanced Federated Learning is a mechanism that provides enhanced security in the FL framework by ensuring that the collaborating parties do not interact at the aggregation time. Another implementation discusses cost-effective FL. As we can understand, communication is a significant bottleneck in the smooth implementation of FL due to multiple devices sending data to the FL manager and the framework requiring multiple iterations to obtain the final model. Cost-effective FL proposes an algorithm to find the best values for the number of iterations and the number of participating clients to get the minimum cost of learning in terms of learning time and energy consumption. This additional work does not hamper the model performance, and the results obtained remain comparable to the ones obtained with plain FL. However, another implementation focuses on measuring the reliability of data owners before aggregating their parameters. This is because the data owners may, intentionally or unintentionally, try to manipulate the global model by sending incorrect updates. Thus this implementation relies on a reputation-based mechanism to calculate the reliability of each participating data owner. It uses consortium blockchain to manage the reputation of each device decentrally. This leads to a more robust and attack-immune FL while preserving plain FL's accuracy.

## 3.3 Importance of FL

Federated Learning is important because of the reasons mentioned above. Some of them are as follows:

- **Privacy:** In contrast to traditional methods where data is sent to a central server for training, federated learning allows for training to occur locally on the edge device, preventing potential data breaches.

- **Data security:** Only the encrypted model updates are shared with the central server, assuring data security.
- **Access to heterogeneous data:** Federated learning guarantees access to data spread across multiple devices, locations, and organizations. It makes it possible to train models on sensitive data, such as financial or healthcare data while maintaining security and privacy.

Federated learning aims to build a joint ML model based on the data located at multiple sites. There are two processes in federated learning: model training and model inference. In the process of model training, information can be exchanged between parties but not the data. The exchange does not reveal any protected private portions of the data at each site. The trained model can reside at one party or shared among multiple parties. At inference time, the model is applied to a new data instance. For example, in a B2B setting, a federated medical-imaging system may receive a new patient who's diagnosis come from different hospitals. In this case, the parties collaborate in making a prediction. Finally, there should be a fair value-distribution mechanism to share the profit gained by the collaborative model. Mechanism design should done in such a way to make the federation sustainable. In broad terms, federated learning is an algorithmic framework for building ML models that can be characterized by the following features, where a model is a function mapping a data instance at some party to an outcome.

- There are two or more parties interested in jointly building an ML model. Each party holds some data that it wishes to contribute to training the model.
- In the model-training process, the data held by each party does not leave that party.
- The model can be transferred in part from one party to another under an encryption scheme, such that other parties cannot re-engineer the data at any given party.
- The performance of the resulting model is a good approximation of ideal model built with all data transferred to a single party.

Federated Learning was proposed in 2016 by Google researchers, that allows remote devices such as smartphones, tablets, PCs or internet of things to collaboratively train a shared model without transmitting raw data across the network. The goal of FL is to optimize a global model while keeping data localized, which can be beneficial for both computational efficiency and data privacy. The minimization problem averaged over N

devices is formulated as minimizing the sum of sum of stochastic functions:

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{N} \sum_{i=0}^{N} f_i(x) := \mathbb{E}_{x_i \sim D_i}[f_i(x; D_i)]$$

Here $f_i$ represents loss, $i$ represents index of client, $D_i$ is the underlying random data distribution. Many algorithms like FedAvg, FedProx, SCAFFOLD, have been developed to tackle optimization challenges in FL.

## 3.4 Challenges in FL

FL has its set of challenges that are different from classic distribution learning regimes. One of the most significant challenges in FL is the large statistical heterogeneity of local data across the devices. This heterogeneity arises mainly due to the non-iid manner of data generalization and collection across the devices. In other words, the data on each device is different from that on other devices, making it difficult to train a global model that performs well on all devices. Another challenge in FL is the partial participation of devices in the network. This is because the number of devices in the network can be massive, and not all devices may be available or willing to participate in the training process. These fundamental challenges make FL highly demanding to tackle, both in terms of optimization algorithm design and in terms of theoretical understanding of convergence behaviour. Such heterogeneity affects the training of models in FL setting. Particularly in FedAvg, statistical heterogeneity introduces client drift due to large number of local updates, which results in bad convergence behaviour of the algorithm as the local updates get too far from the actual global model. This client drift is even compounded by system heterogeneity; thus, performance gets worse.

The **main challenges** that are faced in FL are:

- **System Heterogeneity:** There is significant variability in terms of system characteristics on each device in network because devices in the network may have different computational capabilities, memory constraints, or processing speeds.
- **Statistical Heterogeneity:** Data on each device may come from different sources or have different characteristics, leading to non-iid data across the network.

## 3.5 AGGREGATION ALGORITHMS

In Federated Learning, the aggregation algorithm plays a pivotal role in aggregating the locally trained models from various devices or nodes to forge an updated global model. The primary objective of the aggregation algorithm is to merge the knowledge acquired from individual devices while upholding the sanctity of raw data privacy. A few common aggregation algorithms used in Federated Learning are FedAvg, FedSGD and so on. The ultimate aim of these aggregation algorithms is to attain a global model that reaps the benefits of the knowledge contributed by all participating devices, all the while circumventing the necessity of centrally sharing raw data. By aggregating the local updates, the central server ensures that the global model enhances its overall performance without compromising the privacy of individual data. The selection of the aggregation algorithm hinges upon factors such as the nature of the data, communication constraints, and the desired level of privacy and security.

### 3.5.1 FedAvg

Federated Averaging or FedAvg is a federated optimization technique for non-convex objectives, robust for unbalanced and non-IID data distribution, which is the characteristic of decentralized learning. The technique is based on iterative model averaging. It also considers various factors common to federated learning, such as mobile devices going offline. It averages a subset of the total clients available selected at random. FedAvg is a synchronous, one-shot averaging technique where each client learns a model that minimizes loss on their local data, and these models are averaged by the server to produce the final global model.

Clients can perform more than one local gradient descent update. Instead of sharing the gradients with the central server, weights tuned on the local model are shared. Finally, the server aggregates the clients' weights (model parameters).

**Algorithm 1** *Federated Averaging*. The K clients are indexed by $k$; $B$ is the local minibatch size, $E$ is the number of local epochs, and $\eta$ is the learning rate.

**Server executes:**
initialize $w_0$
**for** each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of $m$ clients)
    **for** each client $k \in S_t$ in parallel **do**
        $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    **end for**
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
**end for**
**function** CLIENTUPDATE($k, w$): // Run on client k
    $B \leftarrow$ (split $P_k$ into batches of size $B$)
    **for** each local epoch $i$ from 1 to $E$ **do**
        **for** each batch $b \in B$ **do**
            $w \leftarrow w - \eta \nabla l(w; b)$
        **end for**
    **end for**
    **return** $w$ to server
**end function**

**For each communication round $t$:**

Client Sampling: The server selects a random subset $S_t$ of clients for the current round. The number of selected clients, $m$, is determined as the maximum of $C \cdot K$ and 1, where $C$ is a constant and $K$ is the total number of clients.

Client Updates (in parallel for each client $k \in S_t$): The server sends the current global model parameter $w_t$ to each client in $S_t$. Each client $k$ performs a local update using the ClientUpdate function. This function runs on each client individually and includes local training on a mini-batch of data. The result is the updated local model parameter $w_{t+1}^k$.

$$w_{t+1}^k = w_t - \eta_k \nabla l(w_t; D_k) \tag{3.1}$$

where $D_k$ represents the local data on client $k$ and $\eta_k$ is the learning rate used for the local update on client $k$.

Aggregation: The server aggregates the local model updates from all clients in $S_t$ to obtain the global model update $w_{t+1}$. The aggregation is a weighted sum, where each client's update is weighted by the ratio of its local dataset size $n_k$ to the total dataset size $m_t$ of all clients in $S_t$.

$$m_t \leftarrow \sum_{k \in S_t} n_k \tag{3.2}$$

$$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k \tag{3.3}$$

For each client $k$, the ClientUpdate function is executed locally. The local training is performed over multiple local epochs, and each epoch involves iterating through mini-batches of the local dataset. The model parameter $w$ is updated using gradient descent with learning rate $\eta$ and the gradient of the loss function $\nabla l(w; b)$ with respect to the mini-batch $b$.

Federated Averaging is a decentralized training approach where clients perform local updates on their data, and the server aggregates these updates to obtain a global model. The algorithm aims to strike a balance between local and global updates, with clients contributing to the global model while preserving data privacy. The client update function captures the essence of local training, updating the model parameters based on the local dataset. Aggregation involves considering the dataset sizes of individual clients, ensuring that each client's contribution is appropriately weighted. Overall, Federated Averaging is a fundamental federated learning algorithm, emphasizing collaborative model training across decentralized devices.

### 3.5.2 FedProx

FedProx, short for Federated Proximal, is a variant of Federated Learning that introduces a regularization term into the federated optimization process. This regularization term is designed to encourage the model updates from individual participants (e.g., devices, clients) to stay close to each other during each round of training. The primary goal of FedProx is to enhance the convergence and stability of Federated Learning, especially when dealing with non-IID data distributions across participants.

FedProx includes a proximal term in the optimization objective. The algorithm is designed for decentralized learning across multiple devices (clients) while incorporating a regularization term to encourage proximity between consecutive model updates. Here's the algorithm:

**Algorithm 2** FedProx (Proposed Framework)

**Require:** $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$

**for** $t = 0, \cdots, T-1$ **do**
  Server selects a subset $S_t$ of $K$ devices at random
  (each device $k$ is chosen with probability $p_k$)
  Server sends $w^t$ to all chosen devices
  **for** each chosen device $k \in S_t$ **do**
    $w_k^{t+1}$ is a $\gamma_k^t$-inexact minimizer of:
    $w_k^{t+1} \approx \arg\min_w h_k\left(w; w^t\right) = F_k(w) + \frac{\mu}{2}\left\|w - w^t\right\|^2$
  **end for**
  **for** each device $k \in S_t$ **do**
    Server receives $w_k^{t+1}$ from device $k$
  **end for**
  Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
**end for**

$K$: Total number of devices (clients).

$T$: Number of communication rounds.

$\mu$: Proximal term coefficient.

$\gamma$: Inexactness parameter.

$w^0$: Initial model parameters.

$N$: Total number of devices in the system.

$p_k$: Probability of selecting device $k$ at each communication round.

Here's a step-by-step explanation of the algorithm: The server randomly selects a subset $S_t$ of $K$ devices based on the specified probabilities $p_k$ for each device. This subset represents the devices that will participate in the current communication round.

The server sends the current global model parameters $w^t$ to all devices in the selected subset $S_t$. Each device $k$ in the selected subset $S_t$ independently performs local computation to find $w_k^{t+1}$, which is a $\gamma_k^t$-inexact minimizer of the local objective function:

$$w_k^{t+1} \approx \arg\min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2} \left\| w - w^t \right\|^2 \tag{3.4}$$

Here, $F_k(w)$ is the local objective function on device $k$. Each device $k$ sends its computed model update $w_k^{t+1}$ back to the server. The server aggregates the received model updates from the selected devices:

$$w^{t+1} = \frac{1}{K} \sum_{k \in S_t} w_k^{t+1} \tag{3.5}$$

This aggregation is a simple average of the model updates from the selected devices. Repeat steps for a total of $T$ communication rounds.

The key components of the FedProx algorithm include the probabilistic selection of devices for each round, the inexact minimization performed by each device, and the aggregation of model updates with a proximal term regularization. This framework allows for collaborative learning across decentralized devices while considering the proximity of consecutive model updates, which can improve convergence and robustness.

### 3.5.3 Scaffold

Scaffold uses control variates, which is for the variance reduction to correct for the client-drift in its local updates. It requires significantly fewer communication rounds and is not affected by data heterogeneity or client sampling. Further, it can take advantage of similarity in the client's data yielding even faster convergence. This quantifys the usefulness of local-steps in distribution optimization.

SCAFFOLD attempts to deal with the client drift with the concept of control variates for server and clients. It determines update direction for global model and for each client and the difference between these update directions is used to shift the local update. In addition, SCAFFOLD has the ability to perform better in the presence of similar clients.

**Algorithm 3** SCAFFOLD: Stochastic Controlled Averaging for Federated Learning

**Server Input:** initial $x$ and $c$, and global step-size $\eta_g$
**Client $i$'s Input:** $c_i$, and local step-size $\eta_l$
**for** each round $r = 1, \ldots, R$ **do**
    sample clients $S \subseteq \{1, \ldots, N\}$
    communicate $(x, c)$ to all clients $i \in S$
    **for** each client $i \in S$ in parallel **do**
        initialize local model $y_i \leftarrow x$
        **for** $k = 1, \ldots, K$ **do**
            compute mini-batch gradient $g_i(y_i)$
            $y_i \leftarrow y_i - \eta_l(g_i(y_i) - c_i + c)$
        **end for**
        $c_i^+ \leftarrow$ (i) $g_i(x)$, or (ii) $c_i - c + \frac{1}{K\eta_l}(x - y_i)$
        communicate $(\Delta y_i, \Delta c_i) \leftarrow (y_i - x, c_i^+ - c_i)$
        $c_i \leftarrow c_i^+$
    **end for**
    $(\Delta x, \Delta c) \leftarrow \frac{1}{|S|}\sum_{i \in S}(\Delta y_i, \Delta c_i)$
    $x \leftarrow x + \eta_g \Delta x$ and $c \leftarrow c + \frac{|S|}{N}\Delta c$
**end for**

$x$: Initial global model parameter.

$c$: Global control parameter.

$\eta_g$: Global step-size.

$c_i$: Local control parameter for client $i$.

$\eta_l$: Local step-size for each client.

$R$: Number of rounds.

$N$: Total number of clients.

$S$: Randomly sampled subset of clients.

$y_i$: Local model parameter for client $i$.

$K$: Number of local iterations.

$g_i(y_i)$: Mini-batch gradient of client $i$'s local model.

$\Delta x$, $\Delta c$: Aggregated updates for the global model parameter and control parameter, respectively.

As seen from algorithm, SCAFFOLD primarily does three things:

1. Local updates at client model.
2. Local updates of local control variate.
3. Aggregation of model update and control variate.

The SCAFFOLD (Stochastic Controlled Averaging for Federated Learning) algorithm is designed for federated learning scenarios, where multiple clients collaborate to train a global model. Explanation of the algorithm goes as: Server has an initial global model parameter $x$, a control variable $c$, and a global step-size $\eta_g$. Each client $i$ has a local control variable $c_i$ and a local step-size $\eta_l$.

**For each communication round $r$:**

Client Sampling: Randomly sample a subset of clients $S \subseteq 1, \ldots, N$.

Server Communication: Communicate the current global model parameter $x$ and the control variable $c$ to all clients $i \in S$.

Client Computation (in parallel for each client $i \in S$): Initialize a local model parameter $y_i$ as a copy of the global model parameter $x$.

Iterate for $K$ local steps: Compute the mini-batch gradient $g_i(y_i)$.

Update the local model:

$$y_i \leftarrow y_i - \eta_l(g_i(y_i) - c_i + c) \tag{3.6}$$

Control Variable Update: Compute a new control variable for each client: $c_i^+$ is updated based on either the gradient at the global model $g_i(x)$ or a difference term.

Communication of Updates: Communicate the updates $(\Delta y_i, \Delta c_i)$ to the server, where:

$$\Delta y_i = y_i - x \quad \text{and} \quad \Delta c_i = c_i^+ - c_i \tag{3.7}$$

Update Local Control Variable: Update the local control variable for each client:

$$c_i \leftarrow c_i^+. \tag{3.8}$$

Global Model and Control Variable Aggregation: Aggregate the updates from all clients:

$$(\Delta x, \Delta c) \leftarrow \frac{1}{|S|} \sum_{i \in S} (\Delta y_i, \Delta c_i) \tag{3.9}$$

Global Model and Control Variable Update: Update the global model parameter and control variable:

$$x \leftarrow x + \eta_g \Delta x \quad \text{and} \quad c \leftarrow c + \frac{|S|}{N} \Delta c \tag{3.10}$$

The algorithm uses stochastic client sampling to select a subset of clients in each round. Each client performs local updates to its model parameters and control variable based on the received global model and control variable. The control variable is updated based on gradients or difference terms, allowing clients to adapt their learning rates. The updates are communicated to the server, which aggregates them and updates the global model and control variable accordingly. This algorithm aims to strike a balance between global and local updates by adaptively controlling the learning rates of individual clients. The control variable mechanism helps in adjusting the step sizes, contributing to better convergence and robustness in federated learning scenarios.

# CHAPTER 4

# MIXTURE DENSITY NETWORKS

## 4.1 Distribution Learning

The dataset is initially partitioned and distributed across various devices within the network. Each device is assigned a specific subset of the entire dataset. Following this, the model parallelism technique is employed, which involves dividing the machine learning model into distinct parts and distributing these parts across different devices. Each device is then responsible for training a specific portion of the model.

Next, we have the task parallelism approach, which entails distributing various tasks related to the machine learning process across nodes. For instance, one node may handle data preprocessing, while others are responsible for model training or evaluation. To initiate the process, the machine learning model is initialized on each node. Model parameters can either be randomly initialized or based on pre-trained models. Each node independently processes its local data and computes gradients.

The general architecture of distribution machine learning follows the parameter server model. In this model, a central parameter server stores the model parameters, while worker nodes compute gradients based on their local data and update the model parameters on the server. The model updates obtained from the computation of gradients, after processing the independent local data of each node, are sent to the central parameter server or combined through other aggregation methods.

The central server then aggregates the model updates from all nodes, adjusts the global model parameters, and shares the updated model with the nodes. The training process iterates through multiple epochs, with nodes continuously updating their local models based on their respective data subsets. The training continues until the model converges to a satisfactory state. However, it is important to note that convergence may take longer due to the asynchronous nature of distribution training.

To ensure efficient exchange of information between nodes, communication and

synchronization are crucial. This includes aggregating model updates, sharing gradients, and synchronizing the overall model state. These processes facilitate effective collaboration and coordination among the distribution nodes, enabling them to collectively improve the model's performance. The distribution machine learning approach involves partitioning and distributing the dataset, dividing the model and tasks across devices and nodes, and utilizing.

The inclusion of fault tolerance mechanisms in frameworks is crucial for ensuring the smooth operation of distribution systems. These systems are inherently prone to failures, and it is essential to have mechanisms in place that allow the training process to continue or recover gracefully in the event of node failures. Additionally, parallel processing plays a vital role in handling the computational load more efficiently.

One widely-used deep learning library, PyTorch, offers support for distribution training through its Distributed Data Parallel (DDP) approach. Distribution learning brings numerous benefits, with scalability being a key advantage. By effectively utilizing distribution computational resources, it enables the scaling up of machine learning tasks. This approach significantly reduces the time required to train complex models on large datasets that may not fit into the memory of a single machine, as it parallelizes the training process. Distributing the workload across multiple nodes enhances resource utilization and accelerates model development. Moreover, the fault tolerance mechanisms present in distribution machine learning (DML) frameworks further enhance the robustness of machine learning systems in distribution environments.

The fault tolerance mechanisms are indispensable in frameworks for distribution systems, ensuring the continuity and resilience of the training process. Parallel processing, exemplified by PyTorch's Distributed Data Parallel approach, optimizes computational load handling. Distribution learning offers scalability, efficiently utilizing distribution computational resources and reducing training time for complex models. By distributing the workload across multiple nodes, resource utilization is improved, and model development is accelerated. The inclusion of fault tolerance mechanisms in DML frameworks enhances the robustness of machine learning systems in distribution environments.

## 4.2 MDNs

Mixture Density Networks (MDNs) are a powerful framework in machine learning that combine the strengths of Neural Networks and Mixture Models. These networks excel in probabilistic regression and uncertainty estimation, providing a comprehensive understanding of uncertainty in predictions. Unlike traditional models that only offer a point estimate, MDNs go beyond by modeling the conditional probability distribution of the target variable. They achieve this by utilizing a mixture of Gaussian distributions, allowing MDNs to capture complex and multimodal relationships between inputs and outputs. The ability of MDNs to estimate uncertainty as a distribution is particularly valuable in domains such as robotics, finance, and medical diagnosis. In these fields, having knowledge of the model's confidence is crucial for decision-making. MDNs serve as a versatile tool, enhancing decision-making and reliability in various fields by quantifying prediction uncertainty.

By blending the power of mixture models with neural networks, MDNs offer a unique advantage in probabilistic regression and uncertainty estimation. These networks leverage the flexibility of neural networks to capture intricate relationships, while also harnessing the probabilistic modeling capacity of mixture models. MDNs estimate the conditional probability distribution of the target variable by employing a mixture of Gaussian distributions. Each component of the mixture represents a distinct outcome or mode, enabling MDNs to capture complex data distributions with multiple modes.

The key differentiator of MDNs lies in their ability to provide a full probability distribution rather than a single point estimate. This approach allows MDNs to offer rich insight into prediction uncertainty, making them invaluable in domains where understanding and managing uncertainty are critical. Mixture Density Networks (MDNs) are a powerful tool that combines the strengths of Neural Networks and Mixture Models. They excel in probabilistic regression and uncertainty estimation, providing a comprehensive understanding of uncertainty in predictions. By quantifying prediction uncertainty through a full probability distribution, MDNs offer valuable insights in various fields, enhancing decision-making and reliability.

A well-performing MDN can be achieved with minimal hyperparameter tuning. In this study, we conducted experiments using a random dataset to explore the impact of

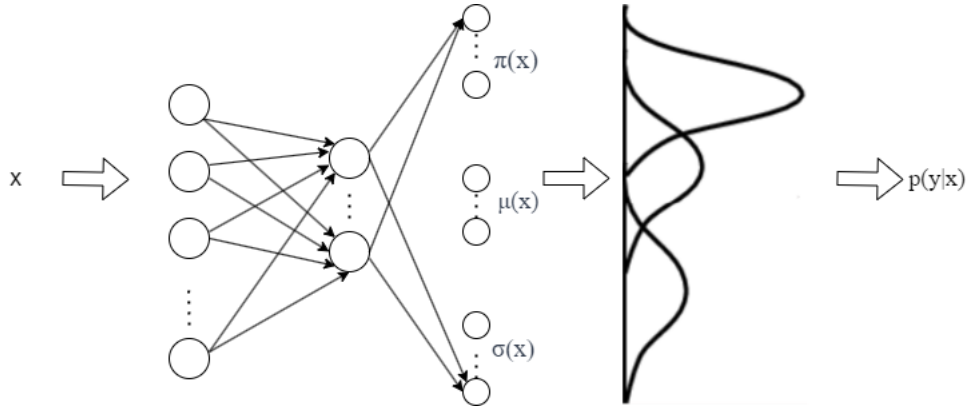


Figure 4.1: Mixture Density Network: The output of a neural network parametrizes a Gaussian mixture model

different parameters on the NN layers of the MDN. Our objective was to first perform regression and then classification tasks. To optimize our MDN model, we carefully defined and adjusted various hyperparameters. To ensure compatibility with our model, the random data was appropriately prepared. While there are multiple models available for fitting random data, we specifically focused on the MDN in this article. The Mixture Density Network (MDN) is a type of neural network that utilizes its output to parameterize a Gaussian mixture model. In essence, the parameters of the mixture model, including the mixing coefficients, means, and variances, are considered integral components of the neural network. By leveraging the capabilities of the MDN, we were able to achieve exceptional results. Through our experiments and careful hyperparameter tuning, we optimized the MDN model to produce the most accurate outcomes.

Bishop uses the Gaussian kernel and explains that any probability density function can be approximated to arbitrary accuracy, provided the mixing coefficients and the Gaussian parameters are correctly chosen. By using the Gaussian kernel in the above equation. it becomes:

$$p(x|\pi,\theta) = \sum_{j=0}^{m-1} \pi_j \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-(\frac{(x-\mu_j)^2}{2\sigma_j^2})} \tag{4.1}$$

Thus the basic structure of MDNs which combine a neural network with a mixture model to capture complex probability distributions goes as:

1. **Neural Network (Encoder/Backbone):** $InputLayer$: The neural network takes in the input data ($X$) and processes it through hidden layers to extract learned features. This part is essentially an encoder or backbone network.

2. **Mixture Model:** The Mixture Model represents the probability density function ($p(x)$) as a mixture of simpler probability distributions. The mixture model is composed of $m$ components (indexed by $j$), each described by a probability density function: $p_j(x \mid \theta_j)$.
   The overall probability density function is modeled as a weighted sum of these components:
   $p(x) = \sum_{j=0}^{m-1} \pi_j \, p_j(x \mid \theta_j)$ where $\pi$ represents the weights or mixing coefficients, and $\theta_j$ are the parameters of the distribution describing the shape and location of each component.

3. **Gaussian Kernel for Mixture Model:** Bishop suggests using the Gaussian kernel to represent each component in the mixture model. The probability density function of the Gaussian component is given by:
   $p(x|\pi, \theta) = \sum_{j=0}^{m-1} \pi_j \, \frac{1}{\sqrt{2\pi\sigma_j^2}} \, e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}$ where $\pi$ are the mixing coefficients, $\mu$ represents the mean parameters, and $\sigma$ represents the standard deviation or variance parameters.
   The mixing coefficients ($\pi$ or $\sigma$) are transformed into probabilities using a Softmax layer to ensure they are between 0 and 1 and sum to unity.

4. **Training:** The entire network, including both the neural network and the mixture model, is trained end-to-end using standard backpropagation. The loss function used for training is the Negative Log Likelihood, which is equivalent to maximizing the Maximum Likelihood Estimation. The goal is to minimize the negative log-likelihood of the observed data under the model.

5. **Inference:** Once trained, the Mixture Density Network can be used for inference. Given new input data, the network calculates the probability distribution $p(x)$ by passing the data through both the neural network and the mixture model.
   The negative log-likelihood is used as a measure of the model's performance during training and as a guide for selecting appropriate model parameters.

# CHAPTER 5

# TRAINING

## 5.1 Baseline

In a centralized system, all the data available is available in one place, and a single global model is trained on the entire dataset where I am taking the batch size of 32. The system then evaluates the model on the test data and gets the testing accuracy. Batch Size is a hyperparameter that defines the number of samples to work through before updating the model parameters. Larger batch sizes can provide a regularizing effect and a faster convergence, but they require more memory. Smaller batch sizes may offer a more stochastic and noisy gradient update but can be computationally more efficient. The batch_size parameter in the DataLoader controls how many samples are processed in each iteration during training. It determines the number of data points fed into the model at once. In our case, with batch_size=32, so the model will process 32 samples in each iteration. This sets the baseline.

## 5.2 Training

The code implements a federated learning approach using a mixture density network. I set the random seeds in the code to ensure reproducibility. This is a good practice, especially when dealing with random initialization and data splitting. MDN model contains necessary components such as pi, sigma, and mu. The probabilities in pi for each sample sum to 1. The loss function used in MDN model is of defined manually in mdn itself, taking into account the probabilities, sigma, mu, and negative log likelihood. The synthetic data generation is done, where three clusters are created, each following normal distribution and concatenating them to form the training data. Then the data is splitting into training and testing sets. Generation of synthetic training data and splitting it into non-IID subsets for each client, and training local MDN models is done. Implementation of Federated Averaging, where local models are averaged to create a

global model, and evaluates the global model. Implementation of Federated Proximal Averaging (FedProx), a variation of federated averaging with a proximal term to penalize the difference between local and global model weights. Implementation Scaffold aggregation, another federated learning approach where models are aggregated using an interpolation parameter (alpha).

The federated averaging implementation aggregates the models correctly by summing the parameters and then dividing by the number of clients, iterating through clients, training local models, and then performing federated averaging. Then evaluating the global model on the test data, which is essential to test the performance of the model. The code ensures that the learning rate of the optimizer associated with the current client during the federated learning process is adjusting so that the learning rate parameter is tuned while training process is going on. The samples are being generated from the testing data using the sample function in MDN class and then calculating the Mean Squared Error (MSE) between these generated samples from the testing data and their true values. While MSE is a valid measure for regression tasks, thus used as a metric for evaluating an MDN for regression.

The number of synchronization rounds is essentially the number of times the global model is updated or aggregated using aggregation algorithm. There are 24 synchronization rounds with 4 local updates per round and a total of 96 epochs happening in the code. Each synchronization round involves updating the global model based on the local models of all clients. After this local training, the global model is updated by aggregating the parameters of all client models using aggregation algorithm like federated averaging, fedprox as well as scaffold i.e., after each synchronization round, one update is sent to global model.

MDN is a neural network model designed for predicting probability distributions, particularly Gaussian Mixture Models. PyTorch, a deep learning library is used to implement this.The code demonstrates federated learning scenarios with three different aggregation methods FEDAVG, FEDPROX, and SCAFFOLD. It uses synthetic data, MDN models, and PyTorch for implementation. Federated learning is a distribution machine learning approach where models are trained locally on separate devices, and the knowledge is aggregated to a global model.

# CHAPTER 6

# RESULTS

## 6.1 Results

### 6.1.1 MDN regression analysis

The dataset utilized for training the models is generated randomly, as is the dataset used for testing the models. We employed a straightforward Mixture Density Network (MDN) with a single nonlinearity, which is trained to produce 1D samples based on a 2D input. By optimizing the hyperparameters, we can obtain a model that achieves the highest accuracy.

Results: Implementation and Findings for Regression using MDN. It is important to note that the hyperparameters may vary, leading to corresponding changes in the achieved accuracies.

| No. of Epochs | Batch Size | Activation function | RMSE | Training Accuracy | Testing Accuracy |
|---|---|---|---|---|---|
| 1000 | 64 | Tanh | 0.3903 | 0.8489 | 0.8478 |
| 1000 | 100 | Tanh | 0.3897 | 0.85099 | 0.8553 |
| 100 | 64 | Tanh | 0.4626 | 0.7857 | 0.7918 |
| 100 | 100 | Tanh | 0.3858 | 0.8505 | 0.8507 |
| 10 | 64 | Tanh | 0.4601 | 0.7877 | 0.7891 |
| 10 | 100 | Tanh | 0.4177 | 0.8267 | 0.8281 |
| 1000 | 64 | ReLU | 0.3869 | 0.8473 | 0.8501 |
| 1000 | 100 | ReLU | 0.3869 | 0.8491 | 0.8481 |
| 100 | 64 | ReLU | 0.3825 | 0.8526 | 0.8493 |
| 100 | 100 | ReLU | 0.4682 | 0.7785 | 0.7833 |
| 10 | 64 | ReLU | 0.4021 | 0.8394 | 0.8364 |
| 10 | 100 | ReLU | 0.4159 | 0.8285 | 0.8258 |

Table 6.1: Test accuracies on learning rate of 0.001

### 6.1.2 Compare FedAvg, FedProx and Scaffold

On applying FedAvg, FedProx, Scaffold algorithm on the random data of normal distribution trained on by the MDN model, varying the number of clients between which the normally distributed data is being divided as well as varying the number of total samples which are distributed among clients, we are able to see following results of testing accuracy for these federated algorithms:

| No. of Samples | No. of Clients | FedAvg | FedProx | Scaffold |
|---|---|---|---|---|
| 40000 | 2 | 0.5180 | 0.3768 | 0.4423 |
| 20000 | 2 | 0.6062 | 0.2161 | 0.2832 |
| 120000 | 3 | 0.5633 | 0.4991 | 0.522 |
| 60000 | 3 | 0.5352 | 0.3721 | 0.3946 |
| 40000 | 3 | 0.5378 | 0.4180 | 0.4680 |
| 20000 | 3 | 0.5021 | 0.4705 | 0.4849 |
| 40000 | 4 | 0.5127 | 0.5854 | 0.5494 |
| 20000 | 4 | 0.5877 | 0.5583 | 0.5371 |
| 120000 | 5 | 0.5275 | 0.5173 | 0.5017 |
| 60000 | 5 | 0.5822 | 0.5769 | 0.5361 |
| 40000 | 5 | 0.5429 | 0.5533 | 0.5232 |
| 20000 | 5 | 0.5569 | 0.5353 | 0.5297 |
| 120000 | 6 | 0.5668 | 0.4972 | 0.5191 |
| 120000 | 10 | 0.5589 | 0.2335 | 0.3217 |
| 60000 | 10 | 0.5579 | 0.2521 | 0.3017 |
| 40000 | 10 | 0.5269 | 0.2126 | 0.2823 |
| 20000 | 10 | 0.5454 | 0.1979 | 0.2286 |

Table 6.2: Test accuracies of FedAvg, FedProx and Scaffold depending on different number of samples and number of clients

Now, I am fixing the number of clients to 5 as we can conclude from the above table that number of clients value at best can be taken as 5 or 6, and tuning the learning rate parameter after every 32 rounds. The whole experiment is performed for different seed values and finally I take the average of all for getting the results visually.

The plots shows the test accuracies of three federated learning algorithms (FedAvg, FedProx, Scaffold) verses number of epochs.

When the total number of data samples are very less:
FedAvg starts with a test accuracy around 55% in the first epoch and shows a general upward trend, reaching a peak around epoch 16. It faces some fluctuations in accuracy during the training process and maintains a relatively stable accuracy level in the later

| | Algorithms | | |
|---|---|---|---|
| | **FedAvg** | **FedProx** | **Scaffold** |
| 1 | 0.5461 | 0.5385 | 0.0355 |
| 2 | 0.5708 | 0.5701 | 0.3867 |
| 3 | 0.5875 | 0.5642 | 0.5185 |
| 4 | 0.5676 | 0.5303 | 0.5691 |
| 5 | 0.5828 | 0.5339 | 0.5995 |
| 6 | 0.5902 | 0.5207 | 0.6240 |
| 7 | 0.5654 | 0.4906 | 0.6315 |
| 8 | 0.5888 | 0.5120 | 0.6288 |
| 9 | 0.5949 | 0.5188 | 0.6292 |
| 10 | 0.5884 | 0.5201 | 0.6345 |
| 11 | 0.5889 | 0.5299 | 0.6375 |
| 12 | 0.5938 | 0.5409 | 0.6403 |
| 13 | 0.5923 | 0.5334 | 0.6443 |
| 14 | 0.5946 | 0.5423 | 0.6482 |
| 15 | 0.5984 | 0.5507 | 0.6435 |
| 16 | 0.6009 | 0.5516 | 0.6472 |
| 17 | 0.6084 | 0.5536 | 0.6420 |
| 18 | 0.6025 | 0.5526 | 0.6480 |
| 19 | 0.6049 | 0.5610 | 0.6523 |
| 20 | 0.6039 | 0.5604 | 0.6497 |
| 21 | 0.6024 | 0.5568 | 0.6514 |
| 22 | 0.6037 | 0.5614 | 0.6494 |
| 23 | 0.6025 | 0.5613 | 0.6539 |
| 24 | 0.6043 | 0.5598 | 0.6520 |

Table 6.3: Test accuracies when total number of samples are 20000 corresponding to 24 synchronization rounds each having 4 local updates

epochs.

FedProx begins with a test accuracy close to 54% in the first epoch and has an initial increase in accuracy up to around epoch 6 and then there's drop in accuracy around epoch 7 and 8. Again, a gradual increase in accuracy in the later epochs, reaching a peak around epoch 16.

Scaffold starts with very poor test accuracy in the first epoch, indicating potential issues, but there is a rapid increase in accuracy in the early epochs itself, surpassing the other algorithms. It keeps on improving and reaches a peak at around epoch 25. It has a relatively consistent and strong performance compared to the other algorithms.

| | **Algorithms** | | |
|---|---|---|---|
| | **FedAvg** | **FedProx** | **Scaffold** |
| 1 | 0.5747 | 0.5680 | 0.3105 |
| 2 | 0.5618 | 0.5289 | 0.5011 |
| 3 | 0.5283 | 0.5056 | 0.5489 |
| 4 | 0.5924 | 0.4209 | 0.5364 |
| 5 | 0.5917 | 0.4334 | 0.5373 |
| 6 | 0.5947 | 0.4176 | 0.5300 |
| 7 | 0.5754 | 0.4275 | 0.5206 |
| 8 | 0.5997 | 0.4180 | 0.5129 |
| 9 | 0.5929 | 0.4062 | 0.5092 |
| 10 | 0.5832 | 0.4084 | 0.5119 |
| 11 | 0.5828 | 0.4211 | 0.5211 |
| 12 | 0.5848 | 0.4295 | 0.5318 |
| 13 | 0.5846 | 0.4271 | 0.5366 |
| 14 | 0.5806 | 0.4423 | 0.5355 |
| 15 | 0.5790 | 0.4503 | 0.5395 |
| 16 | 0.5705 | 0.4867 | 0.5507 |
| 17 | 0.5690 | 0.4903 | 0.5592 |
| 18 | 0.5708 | 0.4920 | 0.5628 |
| 19 | 0.5692 | 0.4904 | 0.5669 |
| 20 | 0.5712 | 0.4915 | 0.5682 |
| 21 | 0.5751 | 0.4933 | 0.5720 |
| 22 | 0.5747 | 0.4943 | 0.5751 |
| 23 | 0.5753 | 0.4941 | 0.5790 |
| 24 | 0.5793 | 0.4960 | 0.5820 |

Table 6.4: Test accuracies when total number of samples are 120000 corresponding to 24 synchronization rounds each having 4 local updates

When the total number of data samples are good in number:
FedAvg starts with an accuracy around 57% and fluctuates around this value for the first few epochs. It then increases and reach till a peak around epoch 8 with an accuracy of

approximately 60%, again shows some fluctuations but generally has an accuracy above 57% for the rest of the epochs.

FedProx begins with an accuracy of 57% and drops significantly in the early epochs, reaching around 42% at epoch 3. There is a visible fluctuating pattern with occasional peaks and valleys. It gradually improves and stabilizes after epoch 12, reaching an accuracy close to 50%.

Scaffold starts with a relatively low accuracy of 31% and quickly improves in the first few epochs. There is a significant increase at around epoch 3 and continues to rise and stabilizes at around an accuracy of 57% towards the end, showing a consistent upward trend.

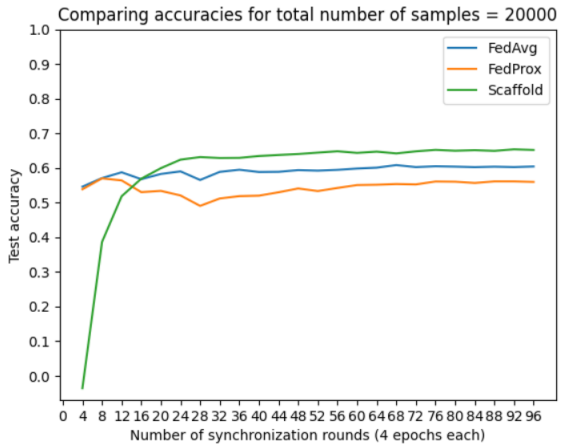


Figure 6.1: Test accuracies vs epochs when total number of data samples are 20000

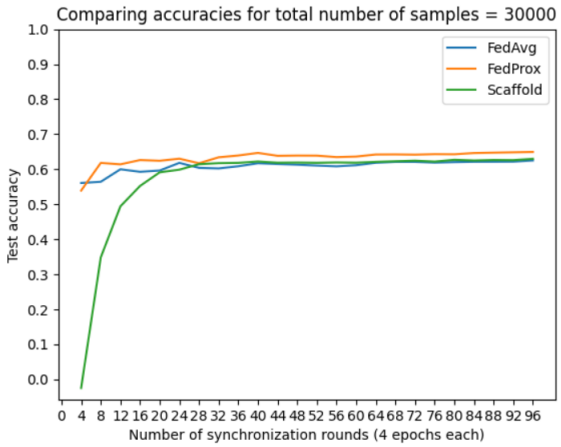


Figure 6.2: Test accuracies vs epochs when total number of data samples are 30000

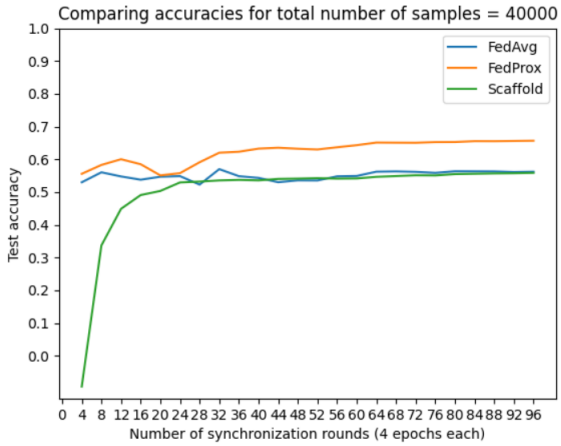


Figure 6.3: Test accuracies vs epochs when total number of data samples are 40000

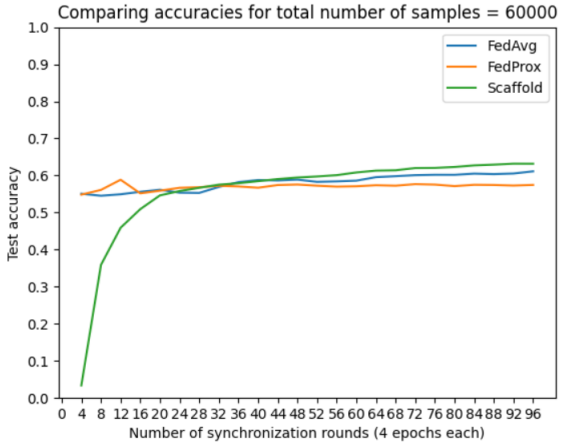


Figure 6.4: Test accuracies vs epochs when total number of data samples are 60000

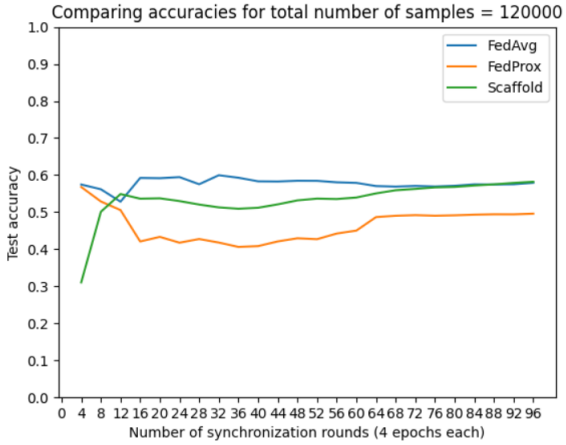


Figure 6.5: Test accuracies vs epochs when total number of data samples are 120000

In conclusion, Scaffold demonstrates a consistent upward trend, starting from a low accuracy and gradually improving, surpassing the other two algorithms in the later epochs, thus has a more consistent and significant improvement over the epochs. FedAvg and FedProx exhibit more variability in their performance, with fluctuations and changes in the trend with FedAvg generally maintaining a higher accuracy than FedProx and a relatively stable performance with a moderate accuracy. The peak accuracies for all algorithms are reached around the same epoch (around epoch 16), suggesting that further training may not lead to significant improvements. This visual representation provides insights into the comparative performance of the algorithms over the training epochs, aiding in the understanding of their learning patterns and convergence behavior. It can guide decisions related to model selection and tuning based on the observed test accuracies.

## 6.2 Conclusion

Federated learning is a distribution system but in a secure way. It allows the application of various machine-learning techniques in a variety of domains. Federated learning is a distribution paradigm that can address security issues while improving the performance of the global model. However, further research is needed in federated learning to know the optimal trade-off between security and efficiency. Off-course privacy is an essential issue in federated learning, and differential privacy techniques ensure data privacy preservation while maintaining the model's efficiency. Moreover, security is also an essential issue of federated learning, and methods such as secure aggregation and secure multi-party computation may ensure the secure training of the model. Overall, federated learning has great potential to transform machine learning by enabling privacy-preserving, decentralized collaboration between devices.

# CHAPTER 7

# Discussion and Future Scope

I explored the application of Mixture Density Networks (MDNs) for regression tasks on datasets following normal distribution. An MDN is a combination of neural networks with the probabilistic modeling strengths of Gaussian Mixture Models (GMM). This unique fusion enables MDNs to effectively capture the underlying patterns and complexities in normally distributed data, making them highly suitable for such datasets.

A notable limitation emerged during the experimentation was that while MDNs perform admirably with normally distributed data, their efficacy significantly diminishes when dealing with data following different distributions. I experimented with poisson distribution data but the results were not as expected. This limitation is primarily due to the inherent structure of the GMM component in the MDN, which is intrinsically designed to model data that adheres to a Gaussian distribution.

To address this challenge, there can be the development of a new model that incorporates a Poisson Mixture Model (PMM) in place of the GMM within the MDN framework. This proposed model aims to extend the applicability of MDNs to datasets that follow a poisson distribution as well. By integrating a PMM, the model will be better equipped to handle datasets with these characteristics, expanding the versatility of MDNs to a broader range of applications.

The neural network and Poisson mixture model combination would function by having the neural network component learn from the input features, similarly to the standard MDN approach. The critical difference lies in the output layer, where the Poisson mixture model would come into play. This model would estimate the parameters of a Poisson distribution (such as the rate parameter) for each component in the mixture, providing a probabilistic output that aligns more closely with the nature of Poisson-distributed data. The development of a hybrid model combining neural networks with a Poisson mixture model represents a promising advancement in the field of probabilistic modeling and regression analysis.

It is our conjecture that with our data distribution, fedprox and scaffold are not carrying their properties which they otherwise do for text data and image data. The data distribution that has been generated in the code may not exactly align with these algorithms, therefore, the conjecture is that their properties don't hold here. This is our future plan to explore more analytically why this is happening.

# REFERENCES


[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data." arXiv, Jan. 26, 2023. Accessed: Jan. 11, 2024. [Online]. Available: http://arxiv.org/abs/1602.05629

[2] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks." arXiv, Apr. 21, 2020. Accessed: Jan. 11, 2024. [Online]. Available: http://arxiv.org/abs/1812.06127

[3] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning." arXiv, Apr. 09, 2021. Accessed: Jan. 11, 2024. [Online]. Available: http://arxiv.org/abs/1910.06378

[4] R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated Learning for Healthcare: Systematic Review and Architecture Proposal," ACM Trans. Intell. Syst. Technol., vol. 13, no. 4, pp. 1–23, Aug. 2022, doi: 10.1145/3501813.

[5] J. Liu et al., "From distributed machine learning to federated learning: a survey," Knowl Inf Syst, vol. 64, no. 4, pp. 885–917, Apr. 2022, doi: 10.1007/s10115-022-01664-x.

[6] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-Effective Federated Learning Design." arXiv, Dec. 15, 2020. Accessed: Apr. 16, 2023. [Online]. Available: http://arxiv.org/abs/2012.08336

[7] C. Ma et al., "When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm." arXiv, Jun. 04, 2021. Accessed: Apr. 16, 2023. [Online]. Available: http://arxiv.org/abs/2009.09338

[8] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and

Applications." arXiv, Feb. 13, 2019. Accessed: Jan. 12, 2024. [Online]. Available: http://arxiv.org/abs/1902.04885

[9] X. Li, W. Yang, K. Huang, S. Wang, and Z. Zhang, "ON THE CONVERGENCE OF FEDAVG ON NON-IID," 2020.

[10] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," IEEE Signal Process. Mag., vol. 37, no. 3, pp. 50–60, May 2020, doi: 10.1109/MSP.2020.2975749.

[11] A. A. Abdellatif et al., "Communication-efficient hierarchical federated learning for IoT heterogeneous systems with imbalanced data," Future Generation Computer Systems, vol. 128, pp. 406–419, Mar. 2022, doi: 10.1016/j.future.2021.10.016.

[12] B. S. Alsulami, C. Bajracharya, and D. B. Rawat, "Game theory-based attack and defense analysis in virtual wireless networks with jammers and eavesdroppers," Digital Communications and Networks, vol. 7, no. 3, pp. 327–334, Aug. 2021, doi: 10.1016/j.dcan.2021.04.002.

[13] Ng D, Lan X, Yao MM, Chan WP, Feng M. Federated learning: a collaborative effort to achieve better medical imaging models for individual sites that have small labelled datasets. Quant Imaging Med Surg. 2021 Feb;11(2):852-857. doi: 10.21037/qims-20-595. PMID: 33532283; PMCID: PMC7779924.

[14] Z. Chen, W. Liao, K. Hua, C. Lu, and W. Yu, "Towards asynchronous federated learning for heterogeneous edge-powered internet of things," Digital Communications and Networks, vol. 7, no. 3, pp. 317–326, Aug. 2021, doi: 10.1016/j.dcan.2021.04.001.

[15] I. Feki, S. Ammar, Y. Kessentini, and K. Muhammad, "Federated learning for COVID-19 screening from Chest X-ray images," Applied Soft Computing, vol. 106, p. 107330, Jul. 2021, doi: 10.1016/j.asoc.2021.107330.

[16] M. Hao, H. Li, X. Luo, G. Xu, H. Yang and S. Liu, "Efficient and Privacy-Enhanced Federated Learning for Industrial Artificial Intelligence," in IEEE Transactions on Industrial Informatics, vol. 16, no. 10, pp. 6532-6542, Oct. 2020, doi: 10.1109/TII.2019.2945367.

[17] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," Journal of Parallel and Distributed Computing, vol. 148, pp. 109–124, Feb. 2021, doi: 10.1016/j.jpdc.2020.10.006.

[18] L. Hu, H. Yan, L. Li, Z. Pan, X. Liu, and Z. Zhang, "MHAT: An efficient model-heterogenous aggregation training scheme for federated learning," Information Sciences, vol. 560, pp. 493–503, Jun. 2021, doi: 10.1016/j.ins.2021.01.046.

[19] W. Huang, T. Li, D. Wang, S. Du, J. Zhang, and T. Huang, "Fairness and accuracy in horizontal federated learning," Information Sciences, vol. 589, pp. 170–185, Apr. 2022, doi: 10.1016/j.ins.2021.12.102.

[20] S. Jia, L. Chen, Y. Chen, B. Li, and W. Liu, "Minimizing the seed set cost for influence spreading with the probabilistic guarantee," Knowledge-Based Systems, vol. 216, p. 106797, Mar. 2021, doi: 10.1016/j.knosys.2021.106797.

[21] C. Jiang, C. Xu, and Y. Zhang, "PFLM: Privacy-preserving federated learning with membership proof," Information Sciences, vol. 576, pp. 288–311, Oct. 2021, doi: 10.1016/j.ins.2021.05.077.

[22] W. Lei, Y. Zhou, and X. Lin, "A physical layer security scheme for full-duplex communication systems with residual self-interference and non-eavesdropping CSI," Digital Communications and Networks, vol. 7, no. 3, pp. 352–361, Aug. 2021, doi: 10.1016/j.dcan.2020.07.004.

[23] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," Computers & Industrial Engineering, vol. 149, p. 106854, Nov. 2020, doi: 10.1016/j.cie.2020.106854.

[24] Kang, Jiawen & Xiong, Zehui & Niyato, Dusit & Zou, Yuze & Zhang, Yang & Guizani, Mohsen. (2019). Reliable Federated Learning for Mobile Networks. IEEE Wireless Communications.

[25] Q. Li, B. He, and D. Song, "Model-Contrastive Federated Learning".

[26] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic Federated Learning". [18] N. Rieke et al., "The future of digital health with federated learning," npj Digit. Med., vol. 3, no. 1, p. 119, Sep. 2020, doi: 10.1038/s41746-020-00323-1.

[27] M. J. Sheller et al., “Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data,” Sci Rep, vol. 10, no. 1, p. 12598, Jul. 2020, doi: 10.1038/s41598-020-69250-1.

[28] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler, and F. Silvestri, “Neural network quantization in federated learning at the edge,” Information Sciences, vol. 575, pp. 417–436, Oct. 2021, doi: 10.1016/j.ins.2021.06.039.

[29] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, “Privacy preservation in federated learning: An insightful survey from the GDPR perspective,” Computers & Security, vol. 110, p. 102402, Nov. 2021, doi: 10.1016/j.cose.2021.102402.

[30] Z. Yang, M. Chen, K.-K. Wong, H. V. Poor, and S. Cui, “Federated Learning for 6G: Applications, Challenges, and Opportunities,” Engineering, vol. 8, pp. 33–41, Jan. 2022, doi: 10.1016/j.eng.2021.12.002.

[31] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” Knowledge-Based Systems, vol. 216, p. 106775, Mar. 2021, doi: 10.1016/j.knosys.2021.106775.