UCLouvain

FACULTE DES SCIENCES

Faculté des sciences

Overview of Federated learning, its challenges and algorithms

Author: **Lavinia Myo Lemegne**
Supervisor: **Eugen Pircalabelu**
Reader: **Johan Segers**
Academic year 2023–2024
Master [120] en science des données, orientation statistique

Acknowledgment

Foremost, I would like to express my gratitude to my promoter, Eugen Pircalabelu, who guided me through this thesis. He was always available with excellent advices, patient, and a meticulous correction of my work that allowed me to apply myself more in work even in moments of uncertainty and produce this document of which I'm proud.

Then, I would like to thank my family, who believe in me and support me in my choices. Their presence and encouragement gave me a great motivation to achieve this goal that I set for myself.

A big thank you to my friends and anyone who, directly or indirectly, participated in the production of this thesis and my entire academic program.

Contents

List of Figures

Abstract

At the time when we are witnessing a great rise in new technologies using data, the protection of the user data becomes imperative. This in various fields such as commerce, education. For this purpose, several new approaches have emerged, particularly one, introduced for the first time by Google in 2016, called **Federated learning**. Federated learning is a recent technology whose primary objective is the acquisition of high quality models while preserving data privacy. In this work, we provide an overview of the field of Federated learning, introducing its parts and types. Then we discuss about the main challenges and complexities associated with the distributed optimization such as heterogeneity of data and propose solutions to deal with them like data normalization. Afterwards, we focus on algorithms of Federated learning existing in the literature such as FedAvg, FedProx and on their main properties. Then, we inspect their implementation as well as the influence of their hyperparameters and models on the performance through simulations and applications on real data under different scenarios and using different metrics like the accuracy.

Introduction

Human-machine interaction has faced turn-around over the past decade. This revolution, where digital technology occupies a large part, is growing and has led to the management of astronomical quantities of data. This flow raised a lot of questions. One of the major aspects of that is confidentiality and security, especially on sensitive or private data. The thesis focuses on a new approach which makes possible to train machine learning models without centralizing the data, called: **Federated learning (FL)**. This method promises to reconcile performance and privacy. Facilitate the collaboration, while sharing information in a complete security, is the core of Federated Learning. By decentralizing information around a set of clients, Federated Learning aims to combine security with collaboration. In fields such as health, technology and finance, data protection is an ongoing question.

The Federated Learning, with its decentralisation outline also raised the issue of heterogeneity of information across the system (localisation, tools, and other specificities). That is the whole point of this study which aims to provide an overview of the concept of federated learning by presenting a few existing algorithms in Federated Learning such as FedAvg, FedProx, FedNova, Scaffold and discuss some of their properties.

This study stands out by analyzing how these algorithms can be computed and their various influence in terms of robustness, scalability, communication in various scenarios and model such as IID and non-IID, both on simulated and real data. We hope to contribute to a better understanding of the concept of Federated learning.

Our work is therefore divided in five chapters. The first chapter will give an overview of the concept of Federated Learning, beginning with a brief history and ending with a presentation of its different types and parts, while highlighting the various challenges linked to its emergence. The second chapter will outline the challenges and complexities of distributed optimisation. It is there that we will define heterogeneity of data, the architecture of the model and the device conditions. The third chapter will focus on the algorithms and methods where we will present in details some algorithms of federated learning set up by researchers with a particular emphasis on the formulation of the objective. In the fourth

chapter, we will explore some useful properties of algorithms presented in the literature. The fifth chapter will be devoted to simulations and the evaluation of the algorithms on data. We will explore there some useful hyperparameters, by generating data from various conditions and run several simulations on it to explore the robustness of the algorithms and finally we will use real data to evaluate their influence on it. At the end of the thesis, we will discuss and conclude our work and then, present future research prospects in the field of Federated learning.

Chapter 1

Overview of Federated Learning

1.1 Brief history of federated learning

Federated learning (FL) is a recent domain of machine learning that has as objective training machine learning models on decentralized data without concentrating it in one location. This implies that just the updated model is securely transmitted to a central server, ensuring the decentralization of training data among the users. The primary objective of this method is to facilitate the acquisition of high-quality models by preserving data privacy. This make sense when the data are sensitive or belong to several parties with various data protection requirements. Here's a brief historical overview of federated learning:

- **2016**: Firstly, FL is introduced by Google with the aim of minimizing the number of communication rounds in a high-quality centralized model, where the data are distributed non-uniformly across a large number of nodes [McMahan et al., 2016].
- **2017**: An implementation adapted for mobile devices is proposed by Google [McMahan et al., 2017b], exploring practical and technical facets of Federated Learning, which encompass strategies for improving communication efficiency among devices [McMahan et al., 2017a].
- **2018**: FL is integrated into the Gboard keyboard on Android devices by Google to improve text suggestions without compromising users' privacy [Yang et al., 2018].
- **2019**: Apple utilized FL to enhance text prediction and automatic correction on iOS devices while preserving user's privacy [Bhowmick et al., 2019].
- **2020-2022**: Federated learning has seen rapid evolution propelled by advancements in cybersecurity and artificial intelligence technologies. Numerous

academic and industrial works are exploring various aspects of FL, such as algorithm improvement, data security and privacy protection.

- **2023**: The field is making significant strides, with numerous studies investigating its applications in diverse domains like medicine and the Internet of Things. Specifically, the fusion of edge computing (optimisation method used which brings calculation and data storage closer to data sources) and AI, culminating in the emergence of edge intelligence, presents a promising avenue. This approach addresses the burgeoning demand for efficient data processing and optimization structures, especially given the vast volumes of data generated at the network's edge [Akhtarshenasa et al., 2023].

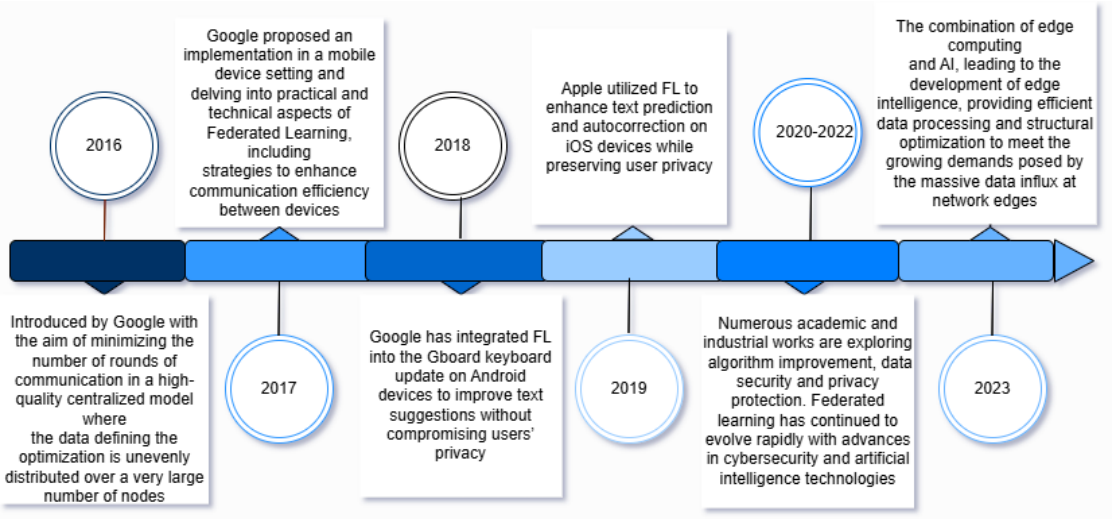


Figure 1.1: History of federated learning

1.2 Challenges in the emergence of the Federated Learning

The emergence of federated learning has been driven by significant challenges. These challenges, especially privacy and security, have demanded the development of innovative approaches to machine learning. First, the creation of many applications is based on the transmission of sensitive and personal data, which presents security risks. Second, the confidentiality of user data is compromised in traditional machine learning models because they require data to be aggregated in a central server, making them attractive targets for hackers.

Also, with the growth of data protection regulations like the GDPR (General Data Protection Regulation) in Europe, companies faced legal challenges in managing user data. Another factor is that Federated Learning permits multiple entities to develop models in collaboration without sharing raw data. For instance hospitals in healthcare applications.

1.3 Parts and types of Federated Learning

Federated learning has emerged because of the compelling need to address the challenges before, ensuring that machine learning can progress responsibly and ethically in an increasingly data-driven world.

1.3.1 Parts

First of all, it is important to talk about the two important parts of Federated Learning : **Clients** and **Server**.

Clients

Considered first elements of Federated Learning, they are the principal beneficiaries of Federated Learning since the knowledge acquired is pitched to improve their experience. They are also called *nodes* and represent the entities that own and often generate the data. Such, they embody the most significant constraints in the process and determine the course of the process as a whole. Their characteristics and configurations are the subject of numerous considerations, especially their number and availability, the quality of their data, their computational and communication capabilities as well as their reliability.

There exist in the literature two federated settings depending on the nature of the clients. One called **cross-device** which refers to scenarios where clients are mobile (smartphones, tablets) and Internet Of things devices (physical objects that are connected to the internet). Only a fraction of them are available at any one time, which makes them unreliable. The second is called **cross-silo** and refers to situations where the clients are organizations or geo-distributed datacenters [Kairouz et al., 2021]. This second one is relevant when organisations with related objectives aim to collectively improve their models through collaboration like hospitals, or when legal constraints prevent a company from centralising its data.

Server

As the second element of Federated Learning, it is also called the *manager or coordinator*. It establishes and maintains the global machine learning model. It is responsible for coordinating the communication between the clients and the server when running the global learning process in a cross-device setting, as it is usually a robust central server and is vital for guaranteeing the stability and reliability of the system. In a cross-silo setting, it is assumed by one of the participating organizations. Its stability and reliability are important as Federated Learning may produce a poor model because the server can fail to provide accurate computation of results [Li et al., 2021a].

1.3.2 Types

After presenting the parts of federated learning, it is also important to present the different types of federated learning frameworks. We distinguish several types of federated learning, including [Mammen, 2021]:

- Federated Transfer Learning: This is similar to traditional Machine Learning, where devices do not share features or the sample space. For example, extending the machine learning model to a larger number of sample instances which are not present in all of the organisations.
- Horizontal Federated Learning: This is used in the case when each device contains datasets with different sample instances but the same feature space. For example merging school datasets can reveal the age group of pupils attending school in a region.
- Vertical Federated Learning: This is applied when each device contains a dataset with different features but share an identical sample space, and these features are complementary. It is necessary to align observations on the different features which can be done using additional techniques like Private Set Intersection [Cristofaro and Tsudik, 2010]. For instance, information on user preferences while maintaining confidentiality may be provided by e-commerce sites to improve the recommendation model on these websites.

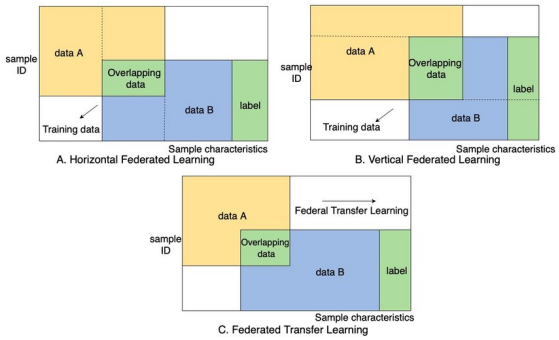


Figure 1.2: Horizontal, Vertical federated learning and federated transfer learning **Source:** Jiang et al. [2020]

1.3.3 Basic Workflow

Here we present a basic workflow for Federated Learning:

1. The client downloads the global model from the server;
2. Client k trains its local data to get a local model;
3. Clients of all parties upload their local model back to the server with the exception of the black client dropping out (for different reasons like lost of internet connectivity) ;
4. After the server receives the model from all parties, it generates a new combined model by an aggregation operation [Yang et al., 2023]. This updated global model serves as the basis for starting the next round of Federated Learning.

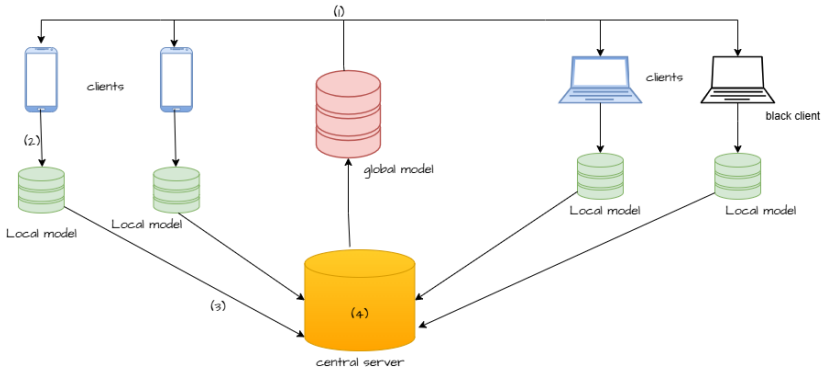


Figure 1.3: Example of workflow of federated learning

Chapter 2

Challenges and Complexities of distributed optimization

The federated learning domain faces specific challenges and complexities coming from the decentralized nature of the data during the learning process. In the following sections, we go through obstacles and complexities in the federated learning domain. The difference of information among client devices represents a considerable challenge in the distributed optimization.

2.1 Heterogeneity of the data

2.1.1 Data distribution

In federated learning, the distribution of the data between devices can be inconsistent and not following the same sampling (non-IID) due to their generation and collection in a non-identically distributed manner across the network. Let's define two clients i, j with a local data distribution $P_i(x, y), P_j(x, y)$ respectively with x, y being the features and labels of the data sample respectively. We distinguish different categories of Non-IID data $P_i(x, y) \neq P_j(x, y)$ in terms of four different skew patterns [Ye et al., 2023a], [Li et al., 2022]:

- **Label skew**: This indicates that the distribution of the labels $P_i(y)$ may differ from one client i to another j. Thus, sampling $q_e \sim dir(\alpha)$ on client i means allocating a proportion $q_{e,i}$ of instances of class e to client i with α the concentration parameter and $dir(\cdot)$, the Dirichlet distribution commonly used and appropriate for the data distribution. For instance, on the keyboards of mobile devices, certain emojis are used differently from one geographical area to another, given their different meanings. Another way of achieving label asymmetry is to define a fixed number of labels $\#lab = l$, then allocate

l different labels to client i and client j, and then distribute the samples of each label randomly and equally between the client, so that client i can have a fixed number of labels l and there is no-overlap between its local data and the sample data of client j.

- **Feature skew**: This indicates that the feature distribution $P_i(x)$ on individual client i may be different from that of client j. For example, in the case of handwritten number recognition, different users can write the same number in different styles.
- **Quality skew**: This indicates that the quality of the data may vary from one client i to another client j. Some may contain data with different degrees of label noise or, due to private data, the collection process may introduce different levels of sample noise.
- **Quantity Skew**: This means that the amount of data may be extremely different between clients i and j. This can happen when part of the data, for certain clients, is not available at certain times in the process for whatever reason like network connectivity. Then, sampling $q \sim dir(\alpha)$ means allocating a proportion q of the total dataset as a local distribution $P_i(x, y)$ with α the concentration parameter and $dir(\cdot)$, the Dirichlet distribution for the data.

2.1.2 Dealing with heterogeneity

In the scenarios where data are heterogeneous, before straightforward aggregation of models a proper preprocessing is mandotory. Therefore, the implementation of data homogenization is imperative.

- **Data Normalization**: Data normalization is a vital preprocessing step in machine learning to ensure that a model is trained on data with comparable statistical properties. Normalization, here, involves scaling the input features to have zero mean and unit variance. By doing so, the model is less likely to be biased towards any specific feature, promoting fair consideration of all inputs during training and that the optimization process converges faster. For example, in a dataset containing both height and weight features, normalization ensures that the variations in one characteristic do not disproportionately affect the model compared to the other. This fosters a more balanced and effective learning process, contributing to improved model performance and generalization on diverse datasets [Goodfellow et al., 2016].
- **Data augmentation**: Data augmentation is a current technique in machine learning aimed at improving the diversity of training samples, thereby ameliorating model robustness and performance. This strategy consist in applying

various transformations to existing data, creating new instances with similar yet different characteristics. For instance, in text classification, data augmentation may include randomly changing the position of a word, replacing a word with its synonym, creating thus a more extensive and varied dataset. By exposing the model to a larger range of inputs through augmented data, it is less subject to overfitting on the original training set and becomes better able to manage real-world variations. Consequently, data augmentation can build more resilient and accurate models that generalize well to unseen data, a crucial aspect in machine learning tasks across domains [de Luca et al., 2022].

2.2 Model architecture

2.2.1 Model heterogeneity

In federated learning, in order to aggregate network parameters of local models into a global one, each client needs to have a local model with the same architecture as the others. However in some cases, there can be a challenge of transferring knowledge between heterogeneous client models in a model-sceptical way in the sense that models may be designed with different structures and then, federated learning can be used without some clients knowing the local model structure of other clients, specially in the collaborative learning between hospitals for example. In this case of healthcare, users may have different devices with different capabilities such as a smartphone or a specialized medical equipment for example. Since the smartphone may have relatively less processing power, using a deep neural network (deepNN) could be suitable in this device. Whereas for specialized medical equipment with a more complex system, a finely tuned model such as recurrent neural network to process a long sequence of data quickly could give high accuracy.

Ye et al. [2023b] gives an overview of this heterogeneity in two orders, firstly when certain clients have the same local model structure as others, federated learning has to be trained for each subset of clients with the same local model structure through clustering algorithms for example and then, communication between inter-cluster must be carried out through special techniques such as knowledge distillation; this is called *partial model heterogeneity*. The second order is called *complete model heterogeneity* and it is used when all clients have different local model structure. In this case, it is necessary to learn a unique local model for each client which can lead to certain costs such as a low communication efficiency and high learning cost but can handle different data distributions.

2.2.2 Model scalability and convergence

Managing large amounts of data from several client devices is not an easy task. In order to achieve it, a crucial technique known as model partitioning can be employed. This approach involves dividing the overall model into manageable parts, allowing each segment to be processed independently on different clients. Distributed processing between these segments ensures efficient coordination, even when the data come from a multitude of heterogeneous sources. This helps the system to process at the same moment datasets while maintaining the consistency of all the models.
Model convergence, particularly in a non-stationary data environment, is a complex challenge. The use of optimisation algorithms like RMSProp (Root Mean Square Propagation) and Adagrad (Adaptive Gradient), is essential to ensure accurate convergence from constantly changing data. These algorithms adjust the model parameters in real time as the data change and allow the model to react quickly to any change that may occur, to adapt instantly to new information, guaranteeing its relevance and effectiveness in a constantly changing context [Mitra et al., 2021].

2.3 Device condition

2.3.1 Device heterogeneity

Sometimes federated learning involves a large number of clients with different capabilities (GPU, memory, battery). Some of these, like mobile devices, may have limited computational resources and experience problems during the process. They can therefore consume a significant amount of time, thus become **stragglers**, unbalancing the system and make it inefficient by causing system bottlenecks or lags. Some of them may become unavailable or fail to participate in the process at certain times, when a client chooses to withdraw or due to network disconnections, forming a *dropout*. This can include states that lead to system loss or failure by introducing instability and uncertainty, as the server may not receive the updates from all the clients, or that can add undesirable biases affecting the accuracy and robustness of the overall model, as it may not fully capture the knowledge of weaker clients.

2.3.2 Stragglers

Stragglers are clients which require a larger amount of time to complete their computations compared to others. They are detected in synchronous settings where the server waits for all the updates of participant devices before continuing to the next round. If we denote by T_i the time required for local computations at a

specified client i in a set of N clients, the total time the server must wait without excluding stragglers is $\mathcal{O}(\max(T_1, T_2, \ldots, T_i, \ldots, T_N))$, which could be significantly long.

To mitigate it impact, a number of approaches have been proposed. The first one is to set the maximum number of computation times on the server, then at the end of the period the clients send their updates and whether or not the number of iterations corresponds to the number requested. Another approach is to use adaptive sampling techniques to choose which clients communicate during each iteration. This strikes a balance between convergence speed and communication overload [Reisizadeh et al., 2022]. Another approach is to interact with a smaller proportion of clients $C < 1$ per round, thereby reducing the total expected waiting time per round $\max(T_1, T_2, \ldots, T_{\lfloor i \times C \rfloor}) \leq \max(T_1, T_2, \ldots, T_i, \ldots, T_N)$ but this slows down the convergence speed as fewer updates are made in the global model in each round and risks introducing bias into the model if the selected clients do not accurately represent the global data distribution.

2.3.3 Fault-Tolerance and Monitoring

In the complex context of federated learning, characterized by the diversity and heterogeneity of the client devices, fault tolerance is an essential element. Hardware or software failures can occur at any time, threatening the continuity of operations. Several strategies are needed to maintain uninterrupted federated learning, even under adverse conditions:

- **Task redundancy**: Task redundancy means replicating key tasks across multiple client devices. So if one fails, the others can take over, ensuring continuity of the learning process [Babasaheb and Raman, 2018]. For instance, in distributed computing environments, if a client device responsible for a specific task fails, redundant copies of that task on other devices can seamlessly take over, preventing disruptions. This redundancy minimizes the impact of device failures and protects against interruptions in scenarios like collaborative machine learning or distributed data processing. By replicating tasks, the system can maintain consistent performance even if an individual device fails, contributing to the reliability and resilience of the entire learning or computing process.

- **Disaster Recovery**: Disaster recovery procedures, such as regular backups and recovery plans for example, ensure that when a major failure occurs, data and operations can be effectively restored [Fan et al., 2021]. For instance, regular backups of critical data and systems are often performed to secure copies of information. In case of natural disasters as an example, these

backups can be used to quickly recover lost data and to continue normal operations. Establishing a comprehensive recovery plans, including strategies for data recovery and system restoration, improves the overall resilience of the IT infrastructure.

Performance challenges are omnipresent in federated learning. Various real-time monitoring techniques enable performance problems to be detected quickly. These include:

- **Real-time monitoring**: Continuous performance monitoring detects early signs of failure or poor performance, and enables rapid intervention. For example, as part of network monitoring, if a server starts to experience an increase in latency or a decrease of debit, the system can be flagged for investigation before a complete failure occurs. Similarly, in manufacturing, real-time monitoring of equipment performance helps to identify a potential problem and to avoid downtime due to maintenance. This proactive approach ensures that corrective actions are taken early and as soon as need, minimising the impact of performance issues and improving overall system reliability.
- **Resource Optimisation**: Continuous optimisation of hardware and software resources ensures efficient use of available capacity, minimising the risk of overload or malfunction [Wang et al., 2021]. For instance, cloud computing platforms dynamically allocate resources based on demand to ensure optimal performance. In a data center, load balancing algorithms distribute tasks across servers to prevent individual components from becoming overwhelmed. This proactive management of resources promotes system reliability, prevents capacity constraints and ensures smooth operations even during periods of maximum use.

2.4 Communication

2.4.1 Communication setting

In federated learning, different clients may have different network environments and different network connectivity settings, for example for mobile devices (3G, 4G, 5G), resulting in inconsistent bandwidth and latency in communication. This heterogeneity increases the cost and complexity of communication, as some devices may need different times to connect and communicate with the server, and therefore connect slowly, while others may be offline. As a result, the client may encounter different degrees of noise and delay during the communication process, reducing its efficiency [Ye et al., 2023a].

2.4.2 Dealing with effective communication

In a Federated Learning environment, inefficient communication can lead to excessive bandwidth usage and increased time required for model training. To ameliorate the efficiency of communication different strategies can be employed :

- **Client Selection Optimization**: A proper selection of client devices participating in each training iteration can reduce unnecessary communication. Criteria like data quality or geographical proximity can be used to select the most relevant clients for each model update. Considering clients geographically close to each other or with high-quality data, communication costs are reduced and then the federated learning process is optimised. For example, in a scenario where the clients are collaborating with each other, the selection of devices with similar data characteristics can lead to more efficient model updates, because these devices contribute with meaningful information to the global model. Thoughtful client selection improves the overall efficiency of federated learning systems by focusing communication efforts on devices that add the most value to the model training process [Smith et al., 2017].
- **Model Compression**: Model compression is a model optimization technique that decreases the size of a model through methods like factorization, quantization, and data compression. Quantization implies reducing the precision of numerical values by making them require less bits for representation. Factorization implies reducing the overall number of parameters by decomposing the weight matrices into smaller fats. Data compression techniques further reduce the size of the model by encoding information more efficiently. For example, a neural network originally using 32-bit floating-point numbers might be quantized to use 8-bit integers. The smaller models resulting from these compression techniques require less data transfer, reducing communication overhead in scenarios such as transmitting models over networks. This is particularly beneficial for applications where bandwidth is limited or where efficient deployment of the model is key [Han et al., 2016].

2.5 Protection of the privacy

2.5.1 Attacks

Because it involves data, federated learning is not exempt from various forms of attack. Some devices can be compromised, which can jeopardise the whole process. According to Rodríguez-Barrosoa et al. [2022], there exist some categories of attacks based on different criteria such as the privacy, the poisoned part of federated learning, the attack moment, etc which are discussed below.

- **The attack moment**: This specifies the time when the attack is carried out and determines the attack's ability to influence the model. We distinguish between attacks at the training time and at the inference time. The first one indicates the phase from data collection and preparation to model training. They have the ability to modify the federated model, which is still being trained, then infer information from the training data. The second one, also known as evasion or exploratory attack, indicates that the attack appears once the model has been trained. Its aim is to produce wrong predictions or to gather information about the characteristics of the model.

- **The poisoned part**: The aim is to corrupt the global learning model by attacking the data or the model. When the attackers have access to the data of one or more clients and can modify it (Data-poisoning), they can modify the labels of a part of the training data by randomly swapping or mixing some of them. They can include patterns in the sample and then associate them with a target class, or normalise the sample and add noise in order to decrease the performance of the model; they can also sample outside the distribution, then use a sample from another domain with the same characteristics or a sample composed of random noise. When directly poisoning model updates sent from the client to the server (Model-poisoning), attackers generate the model weights as a vector of random values of the same dimension as the model weights received from the server, thereby compromising a small portion of the model weights.

- **The privacy**: Despite the fact that the clients never transmit their own private data in federated learning, the models exchanged are subject to the storage of the private training dataset. Attackers can retrieve data (usually images, plain text) from a client participating in the process by exploiting their parameters when they have partial or additional knowledge of the client-side. They can also deduce certain properties of a client or the population that are not supposed to be shared.

2.5.2 Dealing with attacks

Preserving privacy is a significant challenge in federated learning. Therefore, it is crucial to ensure that the data collected from various client devices, cannot be used to identify or reveal specific information about an individual. To ensure this, different mechanisms may be used:

- **Differential privacy**: this means preserving confidentiality by deliberately introducing controlled noise into model updates. The main idea behind this, is to make model updates sufficiently vague so that it is impossible to

distinguish the specific contribution of a particular client. Thereby, instead of directly transmitting exact model updates, a small amount of noise is added to these updates before they are shared or aggregated. This noise is carefully calibrated to preserve the anonymity of individual data by ensuring that the modifications made to the model cannot be traced back to a specific user. This allows organizations to benefit from data without compromising the security and confidentiality of individuals [Yu and Cui, 2023]. Indeed, this cannot provide sufficient privacy guarantee and the analysis of gradient information can still provide private knowledge to the central server. For example, original images from clients can be recovered by obtaining the aggregation gradient returned by the central server.

- **Consent Policies**: They ensure that users understand the purpose of data collection, the types of data collected and with whom it will be shared. This allows users to control their data and give informed consent before their data is being used [Bonawitz et al., 2022]. These are explicit rules and agreements between users and organisations about how data can be collected, used and shared. For example, a data-sharing agreement is a document that defines the conditions for sharing personal data between two or more parties, like the purpose, scope, security and retention of the data.

- **Encryption mechanisms**: In federated learning, classical cryptography protocols such as Secure Multiparty Computation (SMC) has been introduced to protect individual model updates so that the central server is not able to see any local updates, but can still observe the exact aggregated results at each round [Li et al., 2020a]. Other techniques such as Advanced Encryption Standard (AES) ensures that the data are encrypted in transit, which means that even if an unauthorised third party intercepts the data during transfer, it cannot read nor understand it without the appropriate encryption key. Another data masking is the process of replacing sensitive data with fake or scrambled data. This guarantees a high level of data security and preserves the confidentiality of user information [Ma et al., 2022].

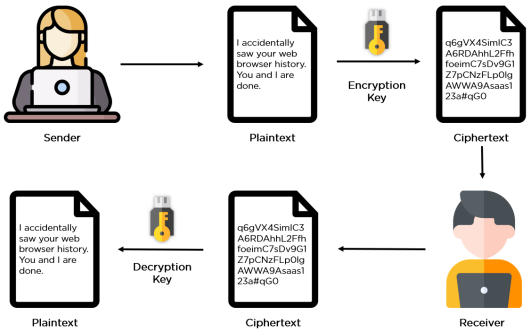


Figure 2.1: Encryption-decryption

Source: https://www.simplilearn.com/data-encryption-methods-article

Chapter 3

Federated Learning algorithms and methods

3.1 Objective formulation

Several algorithms and methods are used in Federated Learning to address various challenges and optimise the collaborative training process across decentralised devices, with a focus on improving confidentiality, communication efficiency, model convergence, and scalability.

We next introduce some notation used throughout the thesis. Let:

- d be the dimensionality or number of features in the model,
- $\mathcal{K} = \{1, 2, ..., N\}$ denotes the collection of all possible clients with N the total number of clients, and $\mathcal{P}$, a distribution supported on it.
- $\forall\ k \in \mathcal{K}$, $D_k = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{n_k}\}$ of size n_k denotes its private data collection, which follows the local data distribution $\mathcal{D}_k$ with $\mathcal{E}_i = \{x_i, y_i\}, \{i = 1, 2, ..., n_k\}$, an input-output (features-label) pair , where $x_i \in \mathbb{R}^d$ represents the feature space and $y_i \in \mathbb{R}$ represents the label space.
- $w = (w_1, w_1, w_2, ..., w_d)^T$ is the parameter vector for the global model.
- $f_k(w; \mathcal{E})$ refers to the loss (mean squared error MSE, cross-entropy loss, ...) of predictions on device k for a sample $\mathcal{E}$ using parameters w.

The goal is to optimize the expected loss [Wang et al., 2021, Yuan et al., 2022, Konečný et al., 2016] as :

$$f(w) := \mathop{\mathbb{E}}_{k \sim \mathcal{P}} \left[\mathop{\mathbb{E}}_{\mathcal{E} \sim \mathcal{D}_k} f_k(w; \mathcal{E}) \right] \tag{1}$$

Using p_k as the relative weight associated to each device k such that:

$$p_k = \frac{n_k}{\sum_{k=1}^{N} n_k}, \text{ with } p_k \geq 0 \text{ and } \sum_{k=1}^{N} p_k = 1 \tag{2}$$

and using sample data available at each client, in practice problem (1) is replaced by the empirical version where:

$$f(w) = \sum_{k=1}^{N} p_k \times F_k(w), \text{such that } F_k(w) = \frac{1}{n_k} \sum_{i=1}^{n_k} f_k(w; \mathcal{E}_i). \tag{3}$$

Therefore the objective in Federated Learning is to:

$$\min_{w \in \mathbb{R}^d} f(w) \equiv \sum_{k=1}^{N} p_k \times F_k(w). \tag{4}$$

Considering the latter (2) in (3) make the objective equivalent to the empirical risk minimization (ERM) objective function of the union of all the local datasets.

In the context of cross-device settings, since the number of device N in (3) can be extremely large, and at a given time only a subset is available to connect to the server, thus participate in the training, the distribution of devices $\mathcal{P}$, the total number of devices N, and thus the total number of device samples $\sum_{k=1}^{N} n_k$ are not known until the start of the training [Wang et al., 2021].

3.2 Processing the optimization problem

In this section, we present an optimization algorithm in order to solve (4). First, we will talk about the framework of adaptive optimization FedOpt (Federated Optimizer), and then we will talk about FedAvg (Federated Averaging), the standard federation optimization method.

3.2.1 Federated Optimization (FedOpt) framework

As a common algorithm to solve (4), FedOpt is an adaptive optimization method where each device performs multiple epochs of training using a *client optimizer* (ClientOPT) to minimise the loss on their local data and the server updates its global model by applying a gradient-based *server optimizer* (ServerOPT) to average client's model updates.

We consider in (1) that each client has a finite collection of data D_k of size n_k, where n_k may vary significantly from device to device, and assume that D_k is divided into a collection of equal-sized batches $\mathcal{B}_k$, each of size B. We denote by $f_k(w; b)$ the average loss on batch b using parameters w with the corresponding gradient $\nabla f_k(w; b)$. and define η_k and η_s as the learning rates associated to the client and the server, respectively, T the number of communication rounds to be performed, and E the number of local epochs we wish to train client k [Reddi et al., 2021].

In algorithm 1, we present in pseudo-code FedOpt framework which allows to estimate w from distributed data across N clients.

Algorithm 1 FedOpt framework

Input: $w^0, T, E, \eta_k, \eta_s, \mathcal{B}_k, \text{ClientOPT}, \text{ServerOPT}$
for each round $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $w_k^{(t,0)} = w^t$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $g_k^t = \nabla f_k(w_k^{(t,e)}; b)$
        $w_k^{(t,e+1)} = \text{ClientOPT}(w_k^{(t,e)}, g_k^t, \eta_k, t)$
      end for
    end for
    $\Delta_k^t = w_k^{(t,E)} - w_k^{(t,0)}$
  end for
  $n_{\text{eff}} = \sum_{k \in S_t} n_k$
  $\Delta^t = \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \Delta_k^t$
  $w^{t+1} = \text{ServerOPT}(w^t, -\Delta^t, \eta_s, t)$
end for

3.2.2 Federated Averaging (FedAvg) Algorithm

The first federated learning algorithm [McMahan et al., 2017b], Federated averaging (FedAvg), is an algorithm for distributed training with a large number of clients. Clients keep their data locally for privacy protection, and a central server is used to communicate between clients. During each round t, the server selects a subset of $K < N$ clients devices by taking a fraction of clients $s_t = \lfloor C \times N \rfloor$ with $N > 0$ as the cardinality of S_t in algorithm 1, C the ratio and distributes the global model weights w^t to all of them. Each client k updates its local model weights with the

received weights, then divides its local training data in order to compute the local update by performing Stochastic Gradient Descent (SGD). Then, the server collects the local updates for each chosen device and generates new global model weights used as starting point for the next communication round. This algorithm is the same as FedOpt framework where ClientOPT, ServerOPT are implicitly Stochastic Gradient Descent (SGD) with a fixed learning rate for the server $\eta_s = 1.0$ [Reddi et al., 2021, Wang et al., 2021].

Optimizing the local objective function of each device using SGD means that if each device computes $\nabla F_k(w)$, the central server updates the parameters as follows:

$$\begin{aligned} w^{t+1} &= w^t - \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \nabla F_k(w^t) \\ &= \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} w_k^t \quad where \quad w_k^t = w^t - \nabla F_k(w^t), \end{aligned} \tag{5}$$

where $\nabla F_k(w^t)$ is the gradient of F_k function from 4 and n_{eff} is the effective sample size given by $n_{\text{eff}} = \sum_{k \in S_t} n_k$.

Assuming that after adding multiple computations in each client, E epochs, the update parameter $w_k^{(t,E)}$ at client k is obtained, the gradient $\nabla F_k(w^t)$ can be approximated by:

$$\nabla F_k(w^t) = w^t - w_k^{(t,E)} \tag{6}$$

The equation (5) becomes:

$$\begin{aligned} w^{t+1} &= w^t - \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \left(w^t - w_k^t \right) \\ &= w^t - \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \left(w^t - w_k^{(t,E)} \right) \\ &= w^t - \sum_{k \in S_t} p_k^{'} \left(w^t - w_k^{(t,E)} \right) \\ &= w^t + \sum_{k \in S_t} p_k^{'} \left(w_k^{(t,E)} - w_k^t \right). \end{aligned} \tag{7}$$

As such, equation (7) uses ServerOPT as being:

$$\begin{aligned} w^{t+1} &= w^t + \sum_{k \in S_t} p_k^{'}(\Delta_k^t) \\ &= w^t + \Delta^t \,, \end{aligned} \tag{8}$$

and ClientOPT as being:

$$w^{(t,e+1)} = w^{(t,e)} - \eta_k g_k^t. \tag{9}$$

The FedAvg algorithm can be written pseudo-code as follows:

Algorithm 2 FedAvg Algorithm

Input: $w^0, T, E, \eta_k, \eta_s = 1, \mathcal{B}_k$
for each round $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $w_k^{(t,0)} = w^t$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $g_k^t = \nabla f_k(w_k^{(t,e)}; b)$
        $w_k^{(t,e+1)} = w_k^{(t,e)} - \eta_k g_k^t$
      end for
    end for
    $\Delta_k^t = w_k^{(t,E)} - w_k^{(t,0)}$
  end for
  $n_{\text{eff}} = \sum_{k \in S_t} n_k$
  $\Delta^t = \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \Delta_k^t$
  $w^{t+1} = w^t + \Delta^t$
end for

Illustration

For illustration, suppose we fixed the fraction $C = 1$, the number of clients $N = 2$, the number of rounds $T = 15$, number of local epochs $E = 20$, a learning rate $\eta_k = 0.001$, size of batches $B = 32$ and the convergence criterion $\epsilon = 1e - 3$ and we want to classify 1000 elements in 2 classes using a Logistic regression model. We obtain the graph in 3.1 comparing the loss between a centralised setting and FedAvg.

Considering the model $y = \frac{1}{1+e^{-(\gamma_0+\gamma_1 x)}}$, with γ_0 being the bias (intercept) of the decision boundary and γ_1 being the coefficient (slope) of the decision boundary that multiplies the features x, we define the loss to be the difference between the prediction and the actual target value ($y_i \in y, i \in \{0, 1, \ldots, n\}$) defined as : $\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$, where:

- y_i is the actual target value (0 or 1).

- $\hat{y}_i = \frac{1}{1+e^{-(\gamma_0+\gamma_1 x_i)}}$ is the predicted probability that the target y_i is 1 .

We see that the loss in centralized decreases more over the rounds compared to the loss when using FedAvg.

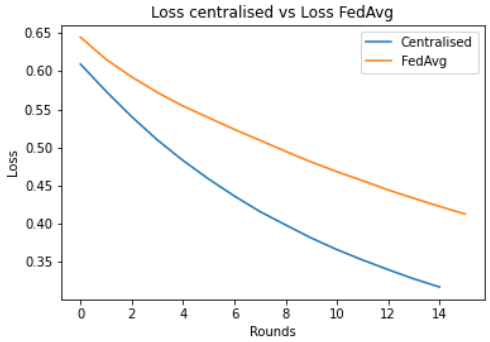


Figure 3.1: Loss FedAvg vs Loss centralised

3.2.3 Discussion about the effects of parameters

There are three parameters (E, B, C) which control the amount of computation [McMahan et al., 2017b]. The two first parameters control the number of local iterations on the data. Each client performs E training passes on its local dataset each round, using a mini-batch of size B. The number of local updates per round (also known as total amount of computation) for each client is given by $\nu_k = E \times \frac{n_k}{B}$ which can raised by increasing n_k or decreasing B.

The third parameter C, the ratio of selected clients, reduces the number of global exchanges required to achieve a fixed accuracy. It is important to strike a balance between selecting too few or too many clients. Sampling a smaller subset can potentially reduce the amount of communication, but also mitigate the straggler effects when clients have heterogeneous computing speeds. It can also influence convergence properties, as too few clients per round can increase the stochastic noise in the training process, see Wang et al. [2021]. On the other hand, because these algorithms require all selected clients to compute their updates before the server aggregates them and moves on to the next communication round, the amount of computation is determined by the slowest clients. To complete a communication round, as we saw in subsection 2.3.1, it either has to wait for the slowest clients to finish or to drop them (method implicitly used in FedAvg), sometimes resulting in sub-optimal convergence [Li et al., 2020b].

3.3 Server Optimizer Algorithms

In this section, we will briefly discuss other ServerOPTs other than SGD (used above in FedAvg) that tune algorithms with the aim of accelerating their convergence. These enable the use of adaptive learning rates without increasing storage and communication costs for the client. Reddi et al. [2021] proposes three server optimizer algorithms which depends in the way the server accumulator s^t is performed at round t and require adding a momentum m^t at the server.

The server accumulator value for these three optimization algorithms with $\beta_1, \beta_2 \in [0, 1]$, momentum parameters, is defined by :

$$s^t = \beta_2 s^{t-1} + (1 - \beta_2)(\Delta^t)^2 \qquad \textbf{FedAdam}$$
$$s^t = s^{t-1} + (\Delta^t)^2 \qquad \textbf{FedAdagrad}$$
$$s^t = s^{t-1} + (1 - \beta_2)(\Delta^t)^2 \text{sign}(s^{t-1} - (\Delta^t)^2) \qquad \textbf{FedYogi}$$

and the momentum add at the server is defined by:

$$m^t = \beta_1 m^{t-1} + (1 - \beta_1)\Delta^t.$$

Thus, ServerOPT becomes :

$$w^{t+1} = w^t + \eta_s \frac{m^t}{\sqrt{s^t} + \epsilon}. \tag{10}$$

To be consistent with equation(8), equation(10) gives :

$$w^{t+1} = w^t + \frac{m^t}{\sqrt{s^t} + \epsilon}. \tag{11}$$

where $\epsilon > 0$ controls the degree of adaptivity of the algorithm and helps to avoid division by 0 in the global update. Zaheer et al. [2018] recommends using $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ in practice. Also, s^t, m are initialized to zero.

In the case of FedAvg, when considering this algorithm as server optimizer, it can be written as :

Algorithm 3 FedAdam, FedAdagrad, FedYogi Algorithms

Input: $w^0, T, E, \eta_k, \eta_s = 1, \mathcal{B}_k, \epsilon, \beta_1, \beta_2, s^{t-1} \geq \epsilon^2, m^{t-1}$
for each round $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $w_k^{(t,0)} = w^t$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $g_k^t = \nabla f_k(w_k^{(t,e)}; b)$
        $w_k^{(t,e+1)} = w_k^{(t,e)} - \eta_k g_k^t$
      end for
    end for
    $\Delta_k^t = w_k^{(t,E)} - w_k^{(t,0)}$
  end for
  $n_{\text{eff}} = \sum_{k \in S_t} n_k$
  $\Delta^t = \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \Delta_k^t$
  $m^t = \beta_1 m^{t-1} + (1 - \beta_1)\Delta^t$
  $s^t = \beta_2 s^{t-1} + (1 - \beta_2)(\Delta^t)^2$ FedAdam
  $s^t = s^{t-1} + (\Delta^t)^2$ FedAdagrad
  $s^t = s^{t-1} + (1 - \beta_2)(\Delta^t)^2 \text{sign}(s^{t-1} - (\Delta^t)^2)$ FedYogi
  $w^{t+1} = w^t + \frac{m^t}{\sqrt{s^t} + \epsilon}$
end for

3.4 Heterogeneity

The heterogeneity of federated learning can be seen in the number of local updates and in the local updates themselves, which leads to *objective inconsistency*, namely the global model converges towards a stationary point of a mismatched objective function which may be arbitrarily different from the true objective.

3.4.1 Federated Proximal FedProx

FedProx is a reparametrised FedAvg algorithm whose objective is to be efficient in a high heterogeneity context and to have a more stable convergence behavior. It reduces the size of the local updates.

During the various iterations, the amount of data acquired by each client is most likely unbalanced and, the local objectives at clients are generally not identical. FedProx introduces a γ-inexact solution factor, a parameter that allows clients to have a variable number of epochs, and carefully merges (even partial) solutions accordingly. To deal with the problem of heterogeneity when only partial information is available, it limits the local updates made by the participating devices to be closer to the initial global model [Li et al., 2020b].

The idea is to add a dynamic regulariser term to the local objectives because excessive local optimization can cause local models to converge to the minima of the local objectives, which can be far away from the global one. This limits the models parameters to remain close to their previous values and ensures that they do not deviate too much from the global model during the training process. This can be achieved by adding into the local objective, a distance metric $\upsilon(\cdot)$ which quantifies the distance between the local model w_k^t and the global model w^t at iteration t as follows:

$$h_k(w_k^t; w^t) = F_k(w_k^t) + \upsilon(w_k^t, w^t). \tag{12}$$

FedProx achieve this by introducing between the local models w_k^t and the global model w^t, a dynamic regulariser called *proximal term* represented by :

$$\upsilon(w_k^t, w^t) = \frac{\mu}{2}||w_k^t - w^t||^2, \mu \geq 0. \tag{13}$$

with μ a hyperparameter controlling the strength of the proximal term and $||\cdot||$ the Euclidean distance.

In addition, FedProx implicitly takes into account variable γ's for different devices and at different iterations, instead of assuming a uniform γ for all devices

throughout the training process. Therefore, we define γ_k^t as the inexact solution factor for device k at iteration t, which measures how much local computation is performed to solve the local sub-problem on device k at the t-th round, and formulate the following definition as in Li et al. [2020b]:

Definition 1 (γ_k^t- inexact solution)**.** For a function $h_k(w; w^t) = F_k(w) + \frac{\mu}{2}||w - w^t||^2$, and $\gamma \in [0, 1]$, we say that w^* is a γ_k^t-inexact solution of $\min_w h_k(w; w^t)$ if $||\nabla h_k(w^*; w^t)|| \leq \gamma_k^t ||\nabla h_k(w^t; w^t)||$, where $\nabla h_k(w; w^t) = \nabla F_k(w) + \mu(w - w^t)$.

We define $\tilde{g}_k^t$ as any *optimizer* used to inexactly minimize $h_k(w; w^t)$. It can be the mini-batch SGD as seen in Li et al. [2020b]. FedProx algorithm can then be written as:

Algorithm 4 FedProx Algorithm

Input: $w^0, T, E, \eta_k, \eta_s = 1, \mathcal{B}_k, \mu$
for each round $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $w_k^{(t,0)} = w^t$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $h_k(w_k^{(t,e)}; w^t) = F_k(w_k^{(t,e)}) + \frac{\mu}{2}||w_k^{(t,e)} - w^t||^2$
        $\tilde{g}_k^t = \arg\min_w \{h_k(w; w^t) \mid \|\nabla h_k(w; w^t)\| \leq \gamma_k^t \|\nabla h_k(w^t; w^t)\|\}$
        $w_k^{(t,e+1)} = w^{(t,e)} - \eta_k \tilde{g}_k^t$
      end for
    end for
    $\Delta_k^t = w_k^{(t,E)} - w_k^{(t,0)}$
  end for
  $n_{\text{eff}} = \sum_{k \in S_t} n_k$
  $\Delta^t = \sum_{k \in S_t} \frac{n_k}{n_{\text{eff}}} \Delta_k^t$
  $w^{t+1} = w^t + \Delta^t$
end for

Parameters

The convergence depends on the value of μ, so it is important to tune it. Li et al. [2020b] says "a large μ may potentially slow the convergence by forcing the updates to be close to the starting point, while a small μ may not make any difference". While automatically tuning the parameter is difficult, in practice, μ can be adaptively chosen based on the current performance. For example, increasing μ when seeing the loss increasing and decreasing μ when seeing the loss decreasing.

3.4.2 Illustration

For illustration, suppose we fixed the fraction $C = 1$, the number of clients $N = 2$, the number of rounds $T = 15$,the number of local epochs $E = 20$, a learning rate $\eta_k = 0.001$, the size of batches $B = 32$ and the convergence criterion $\epsilon = 1e - 3$, $\mu = 0.01$ and we want to classify 1000 elements in 2 classes also using a Logistic regression model defined as in the section 3.2.2. In figure 3.2, we compare the loss between a centralised setting and FedProx where we can wee that the FedProx framework struggles to achieve small losses compared to centralized framework.

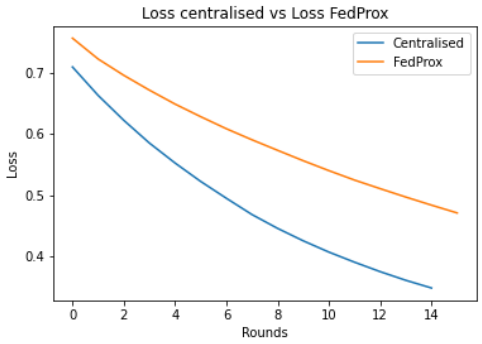


Figure 3.2: Loss FedProx vs Loss centralised

3.4.3 Scaffold

Defined as Stochastic Controlled Averaging for Federated Learning and proposed in Karimireddy et al. [2021], Scaffold is an algorithm which uses controls variates (variance reduction technique used in Monte Carlo methods which consists of exploiting information about the error in the estimated known quantities to reduce the error of the estimated unknown quantities) for the server and clients. It is used to estimate an update direction for each client model and the update direction for the server model in order to reduce client-drift caused by heterogeneous data on FedAvg but doubles the communication size per round. For that, it defines $c^t = \sum_{k \in S_t} c_k^t$ as global control variates (for the server) and c_k^t as control variates for client k (local control variates) at round t. Then, it calculates the drift as $c_k^t - c^t$ and thus, each client performs its local update by using ClientOPT as:

$$w_k^{(t,e+1)} = w_k^{(t,e)} - \eta_k(g_k^t - c_k^t + c^t). \tag{14}$$

where $g_k^t = \nabla f_k(w; \mathcal{E}_k)$.

To update the local control variates, it provides two approaches. The first one, which may be more stable in practise, is to pass over the local data and then,

compute gradients at the global model. The second, which has lower computational complexity is to reuse the previous computed gradients,

$$c_k^{*,t} = \begin{cases} g_k^t, & \text{Approach 1} \\ c_k^t - c^t + \frac{1}{E\eta_k}(w^t - w_k^t), & \text{Approach 2.} \end{cases}$$

The algorithm of Scaffold can be written as :

Algorithm 5 Scaffold Algorithm

Input: $w^0, T, E, \eta_k, \eta_s = 1, \mathcal{B}_k, c^0$
for each round $t = 0, 1, 2, ..T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $c_k^t = c^0$
    $w_k^{(t,0)} = w^t$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $g_k^t = \nabla f_k(w_k^{(t,e)}; b)$
        $w_k^{(t,e+1)} = w_k^{(t,e)} - \eta_k(g_k^t - c_k^t + c^t)$
      end for
    end for
    $\Delta_k^t = w_k^{(t,E)} - w_k^{(t,0)}$
    $c_k^{*,t} = g_k^t$ (i), *or* $c_k^{*,t} = c_k^t - c^t + \frac{1}{E\eta_k}(w^t - w_k^t)$ (ii)
    $\Delta c_k = c_k^{*,t} - c_k^t$
    $c_k^t = c_k^{*,t}$
  end for
  $n_{\text{eff}} = \sum_{k \in S_t} n_k$
  $p_k = \frac{n_k}{n_{\text{eff}}}$
  $\Delta^t = \sum_{k \in S_t} p_k \Delta_k^t$
  $w^{t+1} = w^t + \Delta^t$
  $c^{t+1} = c^t + \sum_{k \in S_t} p_k \Delta c_k$
end for

3.4.4 Federated Normalized Averaging FedNova

FedNova is a generalized algorithm that has been proposed by Wang et al. [2020] to ensure objective consistency by allowing faster clients to perform more local updates than slower clients during each communication round, while preserving fast error convergence. It thus overcomes the problem of unpredictable stragglers

or slow nodes as seen in section 2.3.1. More specifically, the number of local steps in each round is then different from one client to another, so, clients with a greater number of local steps will have a larger local update.

To perform this, FedNova first considers the accumulated gradient to be a linear combination of local gradients:

$$\Delta_k^t = -\eta_s \mathbf{G}_k^t \mathbf{a}_k^t \tag{15}$$

where $\mathbf{G}_k^t = [g_k(w_k^{(t,0)}), g_k(w_k^{(t,1)}), ..., g_k(w_k^{(t,\nu_k)})] \in \mathbb{R}^{d \times \nu_k}$ contains all stochastic gradients defined as $\nabla f_k(w_k^t; b)$at round t of a client k, $\mathbf{a}_k^t \in \mathbb{R}^{\nu_k}$ is a non-negative vector that depends on the client optimizer and defines how stochastic gradients are locally accumulated at round t during ν_k local steps. Then, instead of averaging cumulative local gradients Δ_k^t, the aggregator averages the normalized local gradients denoted as $\mathbf{d_k^t}$ to ensure that the global update is not biased. Thus ServerOPT in equation (8) becomes:

$$\begin{aligned} w^{t+1} &= w^t + \sum_{k \in S_t} p_k^{'} \Delta_k^t \\ &= w^t - \sum_{k \in S_t} p_k^{'} ||\mathbf{a}_k||_1 \times \frac{\eta_s \mathbf{G}_k^t \mathbf{a}_k}{||\mathbf{a}_k||_1} \\ &= w^t - \underbrace{\left(\sum_{k \in S_t} p_k^{'} ||\mathbf{a}_k||_1 \right)}_{\nu_{\text{eff}}\text{: effective } local\ steps} \sum_{k \in S_t} \eta_s \underbrace{\left(\frac{p_k^{'} ||\mathbf{a}_k||_1}{\sum_{k \in S_t} p_k^{'} ||\mathbf{a}_k||_1} \right)}_{\omega_k\text{: } weight} \underbrace{\left(\frac{\mathbf{G}_k^t \mathbf{a}_k}{||\mathbf{a}_k||_1} \right)}_{\mathbf{d_k^t}\text{: } normalized\ gradient}. \end{aligned} \tag{16}$$

Therefore, the update rule in equation (15) becomes:

$$w^{t+1} = w^t - \nu_{\text{eff}} \sum_{k \in S_t} \omega_k \eta_s \mathbf{d_k^t}, \text{ where } \mathbf{d_k^t} = \frac{\mathbf{G}_k^t \mathbf{a}_k}{||\mathbf{a}_k||_1}. \tag{17}$$

and any federated optimization algorithm following the update rule will converge to a stationary point of an surrogate objective function:

$$\tilde{f}(w) = \sum_{k=1}^{N} \omega_k \times F_k(w) \tag{18}$$

In the case where the client optimizer is SGD (FedAvg), $\mathbf{a}_k = [1, 1, ..., 1]^T \in \mathbb{R}^{\nu_k}$, and $||\mathbf{a}_k||_1 = \nu_k$ is the ℓ_1-norm of the vector $\mathbf{a}_k$, thus $\mathbf{d_k^t}$ simply averages gradients over current rounds. So $\nu_{\text{eff}} = \sum_{k \in S_t} p_k^{'} \nu_k$ and $\omega_k = \frac{p_k^{'} \nu_k}{\sum_{k \in S_t} p_k^{'} \nu_k}$ meaning that the normalized gradients with more local steps will be implicitly assigned higher weights.

Setting $\omega_k = p_k^{'}$ makes the algorithm converge to a same stationary point of original objective (4) and the surrogate objective (18). Then, making convergence in terms of true objective (4) by overcoming the problem of objective inconsistency. This, by removing a non-vanishing error $\chi^2_{p\|\omega\|} = \sum_{k\in S_t}(p_i - \omega_k)^2/\omega_k$ (representing the chi-square divergence between vectors p_k and ω_k) becomes 0 in the bounded of the minimal gradient norm defined as :

$$\min_{t\in[T]} \|\nabla F(w^{(t,0)})\|^2 \leq 2 \underbrace{\left[\chi^2_{p\|\omega\|}(\beta^2 - 1) + 1\right]}_{\text{vanishing error term}} \epsilon_{\text{opt}} + \underbrace{2\chi^2_{p\|\omega\|}\kappa^2}_{\text{non-vanishing error due to obj. inconsistency}}$$

Thus, FedNova algorithm can be written as:

Algorithm 6 FedNova Algorithm

Input: $w^0, T, E, \eta_k, \mathcal{B}_k$
for each round $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample a subset $S_t \in \mathcal{K}$ of clients
  for each client $k \in S_t$ **in parallel do**
    $\nu_k = 0$
    $w_k^{(t,0)} = w^{t*}$
    for $e = 0, ..., E$ **do**
      for $b \in \mathcal{B}_k$ **do**
        $g_k^t = \nabla f_k(w_k^{(t,e)}; b)$
        $w_k^{(t,e+1)} = w_k^{(t,e)} - \eta_k g_k^t$
        $\nu_k = \nu_k + 1$
      end for
    end for
    $\mathbf{d_k^t} = \frac{\mathbf{G}_k^t \mathbf{a}_k}{\nu_k}$
  end for
  $n_{\text{eff}} = \sum_{k\in S_t} n_k$
  $p_k^{'} = \frac{n_k}{n_{\text{eff}}}$
  $\nu_{\text{eff}} = \sum_{k\in S_t} p_k^{'} \nu_k$
  $\Delta^t = \sum_{k\in S_t} p_k^{'} \mathbf{d_k^t}$
  $w^{t+1} = w^t - \nu_{\text{eff}} \Delta^t$
end for

Chapter 4

Properties of algorithms

In this chapter, we briefly discuss about the properties of the algorithms presented above. In this part, we fixed that the local step $e \in \{1, ..., E\}$, E the number of local steps.

Considering equation (4), the function F_k may be non convex, having access to an unbiased stochastic gradient g_k of the true client gradient ∇F_k, there are three main properties that distinguish one algorithm from another in order to ensure a convergence to a stationary point of the objective function. Let's first present these properties according to [Karimireddy et al., 2021] paper.

4.1 Assumptions

Assumption 1. [Convex] F_k is said to be a υ-convex, $\upsilon \geq 0$, $\forall k \in \{1, 2, ..., N\}$, if $\forall w, z \in \mathbb{R}^d$,

$$\langle \nabla F_k(w), z - w \rangle \leq - \left(F_k(w) - F_k(z) + \frac{\upsilon}{2} ||w - z||^2 \right).$$

When $\upsilon = 0$, we are in the case of a general convex function as opposed to strongly convex.

Assumption 2. [Bounded Variance] F_k has σ-bounded variance, if $\forall k \in \{1, 2, ..., N\}$, $w \in \mathbb{R}^d$,

$$\mathop{\mathbb{E}}_{\mathcal{E}_k}[||\nabla f_k(w; \mathcal{E}_k) - \nabla F_k(w)||^2] \leq \sigma^2, \sigma^2 \geq 0.$$

This is done only within the same client, not the variance across clients.

Assumption 3. [Lipschitz Gradient smoothness] F_k is said to be L-smooth $\forall k \in \{1, 2, ..., N\}$, if $\forall w, z \in \mathbb{R}^d$,

$$||\nabla F_k(w) - \nabla F_k(z)|| \leq L||w - z||.$$

This implies that F_k is bounded by a quadratic upper bound:

$$F_k(z) \leq F_k(w) + \langle \nabla F_k(w), z - w \rangle + \frac{L}{2}||w - z||^2.$$

If F_k is convex, and w^* is an optimum of f, then:

$$\frac{1}{2LN} \sum_{k=1}^{N} ||\nabla F_k(w) - \nabla F_k(w^*)|| \leq f(w) - f(w^*).$$

Moreover, if f_k is twice-differentiable, assumption (3) implies that:

$$||\nabla^2 F_k(w)|| \leq L \text{ for any } w.$$

Assumption 4. [Bounded gradient dissimilarity] F_k is said to be BG-dissimilar at w, if $\forall w \in \mathbb{R}^d$, there exists $G \geq 0, B \geq 1$ such that ,

$$\frac{1}{N} \sum_{k=1}^{N} ||\nabla F_k(w)||^2 \leq G^2 + B^2||\nabla f(w)||^2.$$

Assumption 5. [B-local dissimilarity] F_k is said to be B-local dissimilar at w, if

$$\frac{1}{N} \sum_{k=1}^{N} ||\nabla F_k(w)||^2 \leq B^2||\nabla f(w)||^2.$$

4.2 Lemma

First, let us present a lemma about norms of random vectors.

Lemma 1 (Relaxed Triangle Inequality)**.** Let us define $\{\mathbf{v}_1, \ldots, \mathbf{v}_\iota\}$ ι vectors in $\mathbb{R}^d$ such that the following are true:

1. $\|\mathbf{v}_i + \mathbf{v}_j\|^2 \leq (1 + a)\|\mathbf{v}_i\|^2 + \left(1 + \frac{1}{a}\right)\|\mathbf{v}_j\|^2$ for any $a > 0$, and
2. $\|\sum_{i=1}^{n} \mathbf{v}_i\|^2 \leq \iota \sum_{i=1}^{n} \|\mathbf{v}_i\|^2$

4.2.1 Heterogeneity

Now, let us talk about bounding heterogeneity by supposing that F_k is convex as in assumption (1), L-smooth as in assumption (3), and BG-dissimilar (4), then :

$$\frac{1}{N}\sum_{k=1}^{N}||\nabla F_k(w)||^2 \leq G^2 + 2LB^2(f(w) - f(w^*))^2.$$

If we suppose now that F_k are convex and L-smooth, then:

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N}\|\nabla F_k(w)\|^2 &\leq \frac{2}{N}\sum_{k=1}^{N}\|\nabla F_k(w^\star)\|^2 + \frac{2}{N}\sum_{k=1}^{N}\|\nabla F_k(w) - \nabla F_k(w^\star)\|^2 \\ &\leq \frac{2}{N}\sum_{k=1}^{N}\|\nabla F_k(w^\star)\|^2 + 4L(f(w) - f(w^\star))\end{aligned}$$

with $G = \frac{2}{N}\sum_{k=1}^{N}\|\nabla F_k(w^\star)\|^2$ when $B^2 = 2$.

However, if we have the assumption (4) and the possibility that the functions may be non-convex, then, according to Sahu et al. [2018] , $G = \varepsilon$, a small tolerance constant, corresponds to local dissimilarity. In the same way, Vaswani et al. [2019] assets that assuming G negligible corresponds to a strong growth condition.

4.3 Rates of convergence, F_k strongly convex

Definition 1. Let $c(x), i(x)$ be two asymptotically non-negative functions such that $c(x) = \tilde{\mathcal{O}}(\mathrm{i(x)})$. This means that $c(x)$ is bounded above by $i(x)$ up to a logarithmic factor. More precisely, there exists constants $a > 0$ such that $c(x) < a \times i(x) \times (\log x)^j$ for a sufficient large x and a constant j [Cormen et al., 2022].

Theorem 4.3.1. [Karimireddy et al., 2021] tells that in the context of strongly convex functions, supposing that F_k satisfies assumptions 2, 3 and 4, then there exist local step-sizes η_k for any $\eta_s \geq 1$, such that the output $\bar{w}^T$ satisfies as:

$$\mathbb{E}[f(\bar{w}^T)] - f(w^*) \leq \tilde{\mathcal{O}}\left(\frac{M^2}{Es_tT\tau} + \frac{LG^2}{\tau^2T^2} + \tau D^2\exp(-\frac{\tau}{16(1+B^2)L}T)\right)$$

with $M^2 := \sigma^2(1+\frac{s_t}{\eta_s^2}) + E(1-\frac{s_t}{N})G^2$, $D := ||w^0 - w^*||^2$, with $\tau \geq 0$, local step-size $\eta_k \leq \frac{1}{8(1+B^2)LE\eta_s}$, the number of communications rounds $T \geq \frac{8(1+B^2)L}{\tau}$ and s_t the cardinality of S_t.

Under the assumptions of theorem 4.3.1 , $\mathbb{E}[f(\bar{w}^T)] - f(w^*)$ has an expected error smaller than ε, a small tolerance constant, if we take into account the fact that: $\eta_s \geq \sqrt{s}_t, \eta_k \leq \frac{1}{(1+B^2)6LE\eta_s}$,

$$T = \tilde{\mathcal{O}}\left(\frac{\sigma^2}{Es_t\varepsilon\tau} + (1 - \frac{s_t}{N})\frac{G^2}{s_t\varepsilon\tau} + \frac{\sqrt{L}G}{\tau\sqrt{\varepsilon}} + \frac{LB^2}{\tau}\right).$$

The same authors Karimireddy et al. [2021] show that the output of SCAFFOLD satisfies for $\tau > 0, \eta_k \leq \min(\frac{1}{81LE\eta_s}, \frac{s_t}{15\tau NE\eta_s})$ and $T \geq \max(\frac{162L}{\tau}, \frac{30N}{s_t})$,

$$\mathbb{E}[f(\bar{w}^T)] - f(w^*) \leq \tilde{\mathcal{O}}\left(\frac{\sigma^2}{Es_tT\tau}(1 + \frac{s_t}{\eta_s^2}) + \frac{N\tau}{s_t}\tilde{D}^2\exp(-\min\{\frac{s_t}{30N}, \frac{\tau}{162L}\}T)\right).$$

where $\tilde{D}^2 := (||w^0 - w^*||^2 + \frac{1}{2NL^2}\sum_{k=1}^N ||c_k^0 - \nabla F_k(w^*)||^2)$.
Considering this, the output has an expected error smaller than ε, using $\eta_s \geq \sqrt{s}_t, \eta_k \leq \min(\frac{1}{81LE\eta_s}, \frac{s_t}{15\tau NE\eta_s})$,

$$T = \tilde{\mathcal{O}}\left(\frac{\sigma^2}{Es_t\varepsilon\tau} + \frac{L}{\tau} + \frac{N}{s_t}\right).$$

Considering the FedProx algorithm, using assumption 5 , with $G = 0$, Li et al. [2020b] define $B(w) = \sqrt{\frac{\mathbb{E}_k[||\nabla F_k(w)||]^2}{||\nabla f(w)||^2}}$ for $||\nabla f(w)|| \neq 0$. When all the local functions F_k are the same, $||\nabla f(w)||^2 = \mathbb{E}_k[||\nabla F_k(w)||]^2$, thus $B(w) = 1$. The weight w is a stationary point that all the local functions F_k agree on. When the sample data on the client devices are homogeneous, $\min_k n_k \to \infty$, then as all the local functions F_k converge to the same expected risk objective function, according to the authors, it follows that $B(w) \to 1$, for every w. However, in a federated setting, the data are often heterogeneous, and because there exist some sampling differences, $B > 1$.

Li et al. [2020b] shows that the dissimilarity defined is assumption 5 is bounded in the sense that, for some $\varepsilon > 0$, there exists B_ε such that $B(w) \leq B_\varepsilon$, for all $w \in S_\varepsilon = \{w \mid ||\nabla f(w)||^2 > \varepsilon\}$. This is justified by the fact that in practical federated settings, the samples are not IID, thus it is reasonable to assume that during the training process, the dissimilarity between local functions remains bounded.

Recall that μ is a hyperparameter controlling the strength of the proximal term.

Theorem 4.3.2. Given some $\varepsilon > 0$, and assuming that for $B \geq B_\varepsilon$, μ, E, γ in algorithm 4 are chosen such that:
$\rho = \left(\frac{1}{\mu} - \frac{\gamma B}{\mu} - \frac{B(1+\gamma)\sqrt{2}}{\bar{\mu}\sqrt{E}} - \frac{LB(1+\gamma)}{\bar{\mu}\mu} - \frac{LB^2(1+\gamma)^2}{2\bar{\mu}^2} - \frac{LB^2(1+\gamma)^2}{\bar{\mu}^2E}(2\sqrt{2E} + 2)\right) > 0$,
with $\bar{\mu} := \mu - L\_ > 0, \exists L\_$ such that $\nabla F_k^2 \succeq -L\_\mathbf{I}$

Then at iteration t of the algorithm 4, we have, recalling that S_t is the set of N devices chosen at round t, the expected decrease in the global objective as follows:

$$\mathbb{E}_{S_t}[f(w^{t+1}] - f(w^t) \leq -\rho||\nabla f(w^t)||^2 \tag{19}$$

Under assumptions of theorem 4.3.2, the authors assert that in addition, after $T = \mathcal{O}(\frac{f(w^0)-f(w^*)}{\rho\varepsilon})$ iterations of FedProx, we have $\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[||\nabla f(w^t)||^2] \leq \varepsilon$.

In cases where F_k are convex functions, and for any k, t, $\gamma_k^t = 0$, if $B \gg 1$ and $B \leq 0.5\sqrt{E}$, we can choose $\mu \approx 6LB^2$ where it follows that $\rho \approx \frac{1}{24LB^2}$. This implies that in order to increase the accuracy using FedProx, we should increase μ appropriately.

Recall that in cases where w is a stationary point that all the local functions agree on, $B(w) = 1$ thus $B = 1$ and $G = 0$, the numbers of communications rounds for FedAvg is $\frac{\sigma^2}{Es_t\varepsilon\tau} + \frac{1}{\tau}$; in the case where all clients have very similar optima and we use full batch gradients, $G = 0, \sigma = 0$, it is $\frac{B^2}{\tau}$, which is the same in FedProx algorithm when $\gamma_k^t = 0$. Now, if we suppose that the clients are not sampling (all of them participate in the training process), the number of rounds for FedAvg becomes $\frac{\sigma^2}{EN\varepsilon\tau} + \frac{G}{\tau\sqrt{\varepsilon}} + \frac{B^2}{\tau}$, the one of FedProx is $\frac{1}{\tau} + \frac{\sigma^2}{\tau^2}$ and the one of Scaffold is of $\frac{\sigma^2}{EN\varepsilon\tau} + \frac{1}{\tau}$.

4.4 Rates of convergence, F_k general convex

Within the setting of general convexity, under the same assumptions as in theorem 4.3.1, there are several changes in the conditions for the output in terms of expected error smaller than ε for FedAvg and Scaffold algorithms. The output for FedAvg satisfies for F_k the assumption 1 for $\varepsilon = 0$, the global step-size $\eta_s \geq \sqrt{s}_t$, the local step-size $\eta_k \leq \frac{1}{(1+B^2)6LE\eta_s}$ and the number of communications rounds $T \geq 1$ is such that,

$$T = \mathcal{O}\left(\frac{\sigma^2 D^2}{Es_t\varepsilon^2} + (1 - \frac{s_t}{N})\frac{G^2D^2}{s_t\varepsilon^2} + \frac{\sqrt{L}G}{\varepsilon^{\frac{3}{2}}} + \frac{LB^2D^2}{\varepsilon}\right)$$

.

However for Scaffold, the output satisfies for $\eta_s \geq \sqrt{s}_t, \eta_k \leq \frac{1}{81LE\eta_s}$,
$T = \tilde{\mathcal{O}}\left(\frac{\sigma^2D^2}{Ks_t\varepsilon^2} + \frac{LD^2}{\varepsilon} + \frac{NF}{s_t}\right)$. with $D := ||w^0 - w^*||^2$, $F := f(w^0) - f(w^*)$.

Here, the number of communication rounds when the clients are not sampling and there is no variance, it is of order $\frac{G}{\varepsilon^{\frac{3}{2}}} + \frac{B^2D^2}{\varepsilon}$, and in Scaffold, of $\frac{D^2}{\varepsilon} + F$. In FedProx, it does not really change a lot except the fact that $\tau = \varepsilon$.

4.4.1 Rates of convergence when F_k is non-convex

Here, non-convex means that F_k does not respect assumption 1. This leads to some changes in the conditions to make the expected error of the output of algorithms

smaller than ε, a small constant. For FedAvg, with $F := f(w^0) - f(w^*)$, the outputs satisfies for $\eta_s \geq \sqrt{s}_t, \eta_k \leq \frac{1}{(1+B^2)6LE\eta_s}$, the number of communications rounds $T = \mathcal{O}\left(\frac{\sigma^2 LF}{Es_t\varepsilon^2} + (1 - \frac{s_t}{N})\frac{G^2F}{s_t\varepsilon^2} + \frac{\sqrt{L}G}{\varepsilon^{\frac{3}{2}}} + \frac{LB^2F}{\varepsilon}\right)$.

For Scaffold, its output satisfies for $\eta_s \geq \sqrt{s}_t, \eta_k \leq \frac{1}{24LE\eta_s}(\frac{s_t}{N})^{\frac{2}{3}}$,

$$T = \mathcal{O}\left(\frac{\sigma^2 LF}{Es_t\varepsilon^2} + (\frac{N}{s_t})^{\frac{2}{3}}\frac{LF}{\varepsilon}\right).$$

When we do not sample the clients ($N = s_t$), the number of rounds for non-convex settings is of order $\frac{F\sigma^2}{Es_t\varepsilon^2} + \frac{F}{\varepsilon}$. Additionally, when the variance is null, the number is of order $(\frac{N}{s_t})^{\frac{2}{3}}\frac{F}{\varepsilon}$ in Scaffold.

Considering FedAdagrad, FedAdam, FedYogi algorithms [Reddi et al., 2021] under assumptions 2, 3, 4 when $\eta_k \leq \frac{1}{16E}\min\{\frac{1}{L}, \frac{1}{T^{\frac{1}{6}}}\left[\frac{\varepsilon}{120L^2G}\right]^{\frac{1}{3}}\}$, $\sigma^2 = \sigma_k^2 + 6K\sigma_s^2$, $\varepsilon = G/L$, with m_t the momemtum parameter at the server, the output satisfies:

$$\min_{0\leq t\leq T-1}\mathbb{E}||f(w^t)]||^2 = \mathcal{O}\left(\frac{F}{\sqrt{m_tET}} + \frac{2\sigma_k^2 L}{G^2\sqrt{m_tET}} + \frac{\sigma^2}{GET} + \frac{\sigma^2 L\sqrt{m_t}}{G^2\sqrt{E}T^{\frac{3}{2}}}\right).$$

where σ_k, σ_s denotes the bounded variance at client k and server s respectively.

When T is sufficiently large compared to E, we obtain a convergence rate of order $\mathcal{O}(\frac{1}{\sqrt{m_tET}})$.

4.5 Sketch of proof of convergence of Scaffold for convex functions

Definition 1. We consider ξ_{t+1} as client-drift which is the deviation of the client from its starting point, and which is defined as:

$$\xi_{t+1} := \frac{1}{EN}\sum_{e=1}^{E}\sum_{k=1}^{N}\mathbb{E}[||w_k^{t+1,e} - w^t||^2].$$

and a control lag $\mathcal{C}_t$ since when sampling the clients, not all the clients control-variate in algorithm 5 get updates at each round:

$$\mathcal{C}_t := \frac{1}{N}\sum_{k=1}^{N}\mathbb{E}||\mathbb{E}[c_k^t] - \nabla F_k(w^*)||^2.$$

We define by $\tilde{\eta} := \eta_s\eta_k E \geq 0$ the effective step-size.

Theorem 4.5.1 (Variance of server update)**.** In any round t, and any $\tilde{\eta} := \eta_s \eta_k E \geq 0$, the variance of the server update is bounded by as:

$$\mathbb{E}[||w^{t+1} - w^t||^2] \leq 8L\tilde{\eta}^2(\mathbb{E}[f(w^t)] - f(w^*)) + 8\tilde{\eta}^2\mathcal{C}_t + 4\tilde{\eta}^2L^2\xi_{t+1} + \frac{12\tilde{\eta}^2\sigma^2}{ES}$$

Remark. An ideal update on client k would be: $w_k \leftarrow w_k + \frac{1}{N}\sum_k g_k$, if we consider that the communication cost is not a concern. Unfortunately such an update requires communicating with all clients for every update step, Scaffold instead uses control variates such that $c_k \approx g_k \quad \text{and} \quad c \approx \frac{1}{N}\sum_j g_k$.

Thus, the local updates of Scaffold remain synchronized and converge for arbitrarily heterogeneous clients, Scaffold in equation 14 simulates the ideal update with $(g_k - c_k + c) \approx \frac{1}{N}\sum_k g_k$.

Let us now show a proof of theorem 4.5.1. Starting from the shared global parameters, $y_k^{(t+1,e)} = w^t$, the local parameters in algorithm 5 are updates in the sense that for any $e \in \{1, ..., E\}$:

$$y_k^{(t,e)} = y_k^{(t,e-1)} - \eta_k(g_k^t - c_k^t + c^t) \text{ with } g_k = \nabla F_k(y_k^{(t,e)}). \tag{20}$$

Then updating the control iterates, with E the number of local steps, using Approach 2 in equation 3.4.3 and remark 4.5 as :

$$\tilde{c}_k^{(t+1)} = c^t - c_k^t + \frac{1}{E\eta_k}(w^t - w_k^{t,E}) = \frac{1}{E}\sum_e g_k. \tag{21}$$

Therefore the local control variate for client k is defined by :

$$c_k^{t+1} = \begin{cases} \tilde{c}_k^{t+1} & \text{if } k \in S_t, \\ c_t^i & \text{otherwise.} \end{cases} \tag{22}$$

The new global parameters are then defined by :

$$w^{t+1} = w^t + \frac{\eta_s}{s_t}\sum_{k \in S^t}\left(w_k^{(t,E)} - w^t\right) \tag{23}$$

and

$$c^{t+1} = \frac{1}{N}\sum_{k=1}^{N} c_k^{t+1} = \frac{1}{N}\left(\sum_{k \in S^t} c_k^{t+1} + \sum_{j \notin S^t} c_j^t\right). \tag{24}$$

Finally, for some weights Ω_t, we output:

$$\bar{w}^T = w^t \text{ with probability } \frac{\Omega_t}{\sum_{t=1}\Omega_t} \text{ for } t \in \{0, ...T-1\}.$$

According to Karimireddy et al. [2021], putting together equations (20) - (27), one may bound at any round t, the variance of the server which can be written as (removing the superscript t everywhere): $\mathbb{E}\|\Delta w\|^2 = \mathbb{E}\left[\frac{\eta_g}{Es_t}\sum_{k\in S_t}(g_k + c - c_k)\right]^2$, which can then be expanded as:

$$\begin{aligned}
\mathbb{E}\|\Delta w\|^2 &\leq \mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k\in S_t}(g_k + c - c_k)\right\|^2 \\
&\leq 4\mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k\in S_t}(g_k - \nabla F_k(w))\right\|^2 + 4\tilde{\eta}^2\mathbb{E}\|c\|^2 + 4\mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k\in S_t}\nabla F_k(w^*) - c_k\right\|^2 \\
&\leq 4\mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k\in S_t}(g_k - \nabla F_k(w))\right\|^2 + 4\tilde{\eta}^2\mathbb{E}\|c\|^2 + 4\mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k\in S_t}\nabla F_k(w^*) - c_k\right\|^2 \\
&\quad + 8L\tilde{\eta}^2\left(\mathbb{E}\left[f(w) - f(w^*)\right]\right) \\
&\leq 4\mathbb{E}\left\|\frac{\tilde{\eta}}{Es_t}\sum_{k,e\in S_t}\left(\nabla F_k(y_k^{(t,e-1)}) - \nabla F_k(w)\right)\right\|^2 + 4\tilde{\eta}^2||\mathbb{E}[c]||^2 \\
&\quad + 4\mathbb{E}\left\|\frac{\tilde{\eta}}{s_t}\sum_{k\in S_t}\nabla F_k(w^*) - ||\mathbb{E}[c_k]||\right\|^2 + 8L\tilde{\eta}^2\left(\mathbb{E}\left[f(w)\right] - f(w^*)\right) + \frac{12\tilde{\eta}^2\sigma^2}{Es_t}.
\end{aligned}$$

Considering lemma 1, we can simplify this bound as :

$$\begin{aligned}
\mathbb{E}\|\Delta w\|^2 &\leq \frac{4\tilde{\eta}^2}{EN}\sum_{k,e}\mathbb{E}\left\|\nabla F_k(y_k^{(t,e-1)}) - \nabla f_k(w)\right\|^2 + 4\tilde{\eta}^2||\mathbb{E}[c]||^2 + 4\frac{\tilde{\eta}^2}{N}\sum_k||\nabla F_k(w^*) - \mathbb{E}[c_k]||^2 \\
&\quad + 8L\tilde{\eta}^2\left(\mathbb{E}\left[f(w)\right] - f(w^*)\right) + \frac{12\tilde{\eta}^2}{Es_t} \\
&\leq \frac{4\tilde{\eta}^2}{EN}\sum_{k,e}\mathbb{E}\left\|\nabla F_k(y_k^{(t,e-1)}) - \nabla F_k(w)\right\|^2 + \frac{8\tilde{\eta}^2}{N}\sum_k||\nabla F_k(w^*) - \mathbb{E}[c_k]||^2 \\
&\quad + 8L\tilde{\eta}^2\left(\mathbb{E}\left[f(w)\right] - f(w^*)\right) + \frac{12\tilde{\eta}^2}{Es_t}.
\end{aligned}$$

since $c = \frac{1}{N}\sum_k c_k$ Since the gradients F_k are B-Lipschitz, then

$$\frac{4\tilde{\eta}^2}{EN}\sum_{k,e}\mathbb{E}\left\|\nabla F_k(y_k^{(t,e-1)}) - \nabla F_k(w)\right\|^2 \leq \frac{4L^2\tilde{\eta}^2\sigma^2}{EN}\sum_{k,e}\mathbb{E}\left\|\nabla y_k^{(t,e-1)} - w\right\|^2 = 4L^2\tilde{\eta}^2\xi \tag{25}$$

and the control variate is

$$\mathcal{C}_t := \frac{1}{N}\sum_{k=1}^{N}\mathbb{E}||\mathbb{E}[c_k] - \nabla F_k(w^*)||^2. \tag{26}$$

Thus at any round t and any $\tilde{\eta} := \eta_s \eta_k E \geq 0$, the variance of the server update, the control lag in Scaffold are respectively bounded by :

$$\mathbb{E}[||w^{t+1} - w^t||^2] \leq 8L\tilde{\eta}^2(\mathbb{E}[f(w^t)] - f(w^*)) + 8\tilde{\eta}^2\mathcal{C}_t + 4\tilde{\eta}^2L^2\xi_{t+1} + \frac{12\tilde{\eta}^2\sigma^2}{ES}.$$

Theorem 4.5.2 (Control lag update)**.** In any round t, and any $\tilde{\eta} := \eta_s\eta_k E \geq 0$, the variance of the server update can be written as:

$$c^{t+1} = \frac{1}{N}\sum_{k=1}^{N} c_k^{t+1} = \frac{1}{N}\left(\sum_{k\in S^t} c_k^{t+1} + \sum_{j\notin S^t} c_j^t\right). \tag{27}$$

Recall that after round t, the control update rule 21 implies that c_k^t is set as per

$$c_k^t = \begin{cases} c_i^{r-1} & \text{if } k \notin S^t \text{ i.e. with probability } \left(1 - \frac{s_t}{N}\right), \\ \frac{1}{E}\sum_{e=1}^{E} g_k & \text{with probability } \frac{s_t}{N}. \end{cases}$$

Taking expectations on both sides yields

$$\mathbb{E}[c_k^t] = \left(1 - \frac{s_t}{N}\right)\mathbb{E}[c_k^{t-1}] + \frac{s_t}{EN}\sum_{e=1}^{E}\mathbb{E}[\nabla F_k(y_k^{(t,e-1)})], \quad \forall k \in [N].$$

In addition for the above expression in the definition 1 of $\mathcal{C}_t$, we get

$$\mathcal{C}_t = \frac{1}{N}\sum_{k=1}^{N}\|\mathbb{E}[c_k^t] - \nabla F_k(w^*)\|^2 \tag{28}$$

$$= \frac{1}{N}\sum_{k=1}^{N}\left\|\left(1 - \frac{s_t}{N}\right)(\mathbb{E}[c_i^{t-1}] - \nabla F_k(w^*)) + \frac{s_t}{EN}\sum_{e=1}^{E}\mathbb{E}[\nabla F_k(y_k^{(t,e-1)})] - \nabla F_k(w^*)\right\|^2 \tag{29}$$

$$\leq \left(1 - \frac{s_t}{N}\right)^2 C_{t-1} + \frac{s_t}{N^2E}\sum_{e=1}^{E}\mathbb{E}\|\nabla F_k(y_k^{(t,e-1)}) - \nabla F_k(w^*)\|^2. \tag{30}$$

Furthermore, we can then simplify using the relaxed triangle inequality 1 as:

$$\begin{aligned}
\mathbb{E}_{t-1}[C_t] &\leq \left(1 - \frac{s_t}{N}\right)C_{t-1} + \frac{s_t}{N^2E}\sum_{k,e}\mathbb{E}\|\nabla F_k(y_k^{(t,e-1)}) - \nabla F_k(w^*)\|^2 \\
&\leq \left(1 - \frac{s_t}{N}\right)C_{t-1} + \frac{2s_t}{N^2}\sum_{k}\mathbb{E}\|\nabla F_k(w^{t-1}) - f(w^*)\|^2 \\
&\quad + \frac{2s_t}{N^2E}\sum_{k,e}\mathbb{E}\|\nabla F_k(y_k^{(t,e-1)}) - F_k(w^{t-1})\|^2 \\
&\leq \left(1 - \frac{s_t}{N}\right)C_{t-1} + \frac{2s_t}{N^2}\sum_{k}\mathbb{E}\|\nabla F_k(w^{t-1}) - F_k(w^*)\|^2 + \frac{2s_t}{N^2E}L^2\sum_{k,e}\mathbb{E}\|y_k^{(t,e-1)} - w^{t-1}\|^2 \\
&\leq \left(1 - \frac{s_t}{N}\right)C_{t-1} + \frac{s_t}{N}(4L(\mathbb{E}[f(w^{t-1})] - f(w^*)) + L^2\xi_t).
\end{aligned}$$

The last two inequalities are obtained by the smoothness of $\{F_k\}$ and the definition 1.

Thus, with $\tilde{\eta} \in [0, 1/L]$:

$$\mathcal{C}_{t+1} \leq (1 - \frac{s_t}{N})\mathcal{C}_t + \frac{s_t}{N}(4L(\mathbb{E}[f(w^t)] - f(w^*)) + 2L^2\xi_{t+1}).$$

Chapter 5

Simulation study

5.1 Tools

In this section, we describe briefly at the tools we use:

- We use for experiments a standard machine having Intel i7-1355U CPU, and 32 Gb RAM,
- Flower version 1.7.0, an open source framework for federated-learning which is customisable and extendable [1]. This means that Flower users can choose different configurations depending on their purpose to do, many of Flower's components can be extended, its code is understandable and a variety of machine learning algorithms can be used.

5.2 Exploration tuning

5.2.1 Scenarios

In this chapter, in order to evaluate the efficiency of most of the algorithms presented in chapter 3, we will perform simulations on different scenarios and analyse the algorithms under those configurations. For that, we will first generate a dataset, then in order to represent a real situation where each client has its own data, we decided to split the dataset over the number of clients under the scenarios.

- The first scenario consists of analysing the algorithms when the data are independent and identically distributed (IID) between the clients.

[1] `https://github.com/adap/flower`

- The second scenario consists of analysing the algorithms when the data are non-IID between the clients. More specifically, we distinguish here three designs:
  - label imbalance based on quantity ($\#lab = l$) where each client has the same quantity of class labels l.
  - label imbalance based on a distribution ($q_e \sim distr(\alpha)$) where the distribution of class labels from the target variable across the clients is different and follow a Dirichlet distribution. More precisely, each client has a different proportion of instances of labels from the target variable.
  - data distribution imbalance ($q \sim dir(\alpha)$) where the data across different client follows a Dirichlet distribution. More precisely, each client has a different quantity of data but the same number of class labels from the target variable. This means that some clients may have more data than others.

5.2.2 Metrics of evaluation

To evaluate the algorithms, we will use the accuracy metric defined as: $accuracy = \frac{\mathsf{TP}+\mathsf{TN}}{\mathsf{TP}+\mathsf{TN}+\mathsf{FP}+\mathsf{FN}}$

where TP is the number of true positives results, TN the number of all samples that should have been identified as negatives, FP the number of false positives and FN the number of false negatives.

5.2.3 Model

We choose to use a mutli-layer perceptron model (MLP) for this classification task. MLP is a type of feed-forward neural network composed of fully connected neurons with a non-linear activation function. In our experiment, we use a MLP with two hidden layers of 64 and 15 hidden units respectively.

In order to explore how the algorithms perform in a classification problem when changing different parameters, we decided to tune them. For that, we generate using *scikit learn*[2] a dataset comprising 15 numerical features, a target class of 3 labels and 10000 records. We opted to split the data between the training set (80%), and the test set (20%). We implemented the server-side evaluation approach, which involves using the test set to evaluate the model. The distribution of the target class in the dataset can be see in figure 5.1 below :

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html`

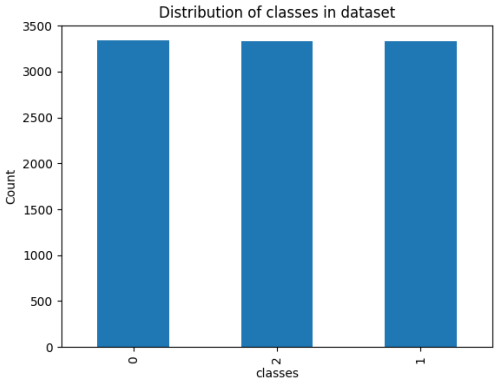


Figure 5.1: Distribution of the target variable

In our experiments, we take as the base $N = 30$ clients, participating in each round, a client learning rate $\eta_k = 0.01$, a Stochastic Gradient Descent as optimizer, a batch size $B = 8$ and a momentum $m = 0.9$. We use $T = 60$ rounds and a number of local epochs of $E = 20$. In the following, all the algorithms have been studied on this basis.

5.2.4 Representation of the scenarios

Before experimenting with the federated learning algorithms, let's analyse the data according to the different scenarios. The main aim is to show the effect of each of these scenarios on the simulated dataset. Here we considered a number $N = 10$ devices in order to have a clear picture of the scenario, and then plotted the concentration of the learning data when there is a case of non-IID label quantity imbalance (figures A.2, 5.2). We see that in the IID, each client receives a different number of records, separated into 2 or 3 labels. For example, client 2 receives 444 records for label 2 and 337 records for label 1. The figures for the case of IID scenario can be see in figures A.1, A.4a) and for the other settings when it is a case of non-IID data in figure A.3 (label distribution imbalance) and figure A.4b (the quantity of data is imbalance).

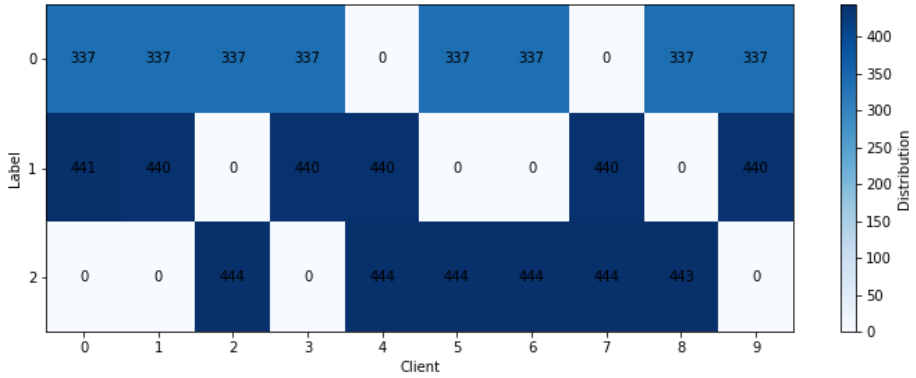
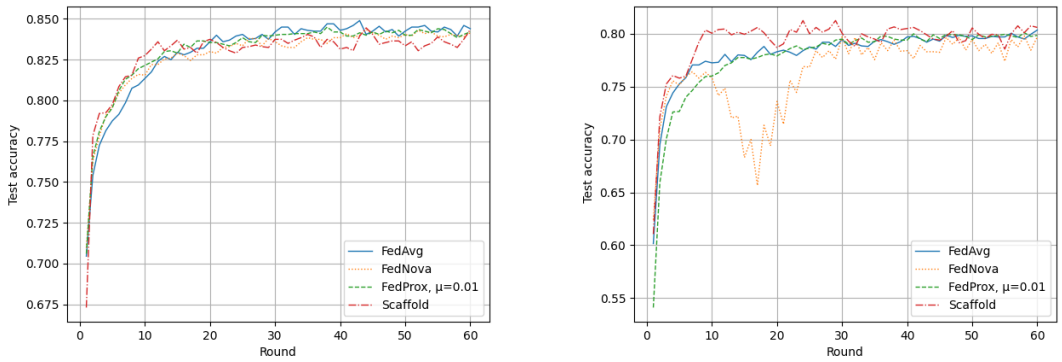


Figure 5.2: Quantity-based label imbalance, 2 labels per client

5.2.5 Effective communication

From figure 5.3b, it can be seen firstly that considering the scenario where the labels distribution is imbalanced, Scaffold and FedNova are unstable while FedAvg and FedProx appear to be similar. Secondly, for the label imbalance based on quantity (figure A.5c, Scaffold becomes more stable while FedAvg and FedProx are more different. On the other hand, for the data distribution imbalance, FedAvg, FedProx and FedNova seem to be close. Furthermore, the fact that FedAvg and FedProx are similar in the scenarios throughout the process is certainly due to the influence of the choice of μ, the regularisation term in FedProx which is small, meaning that the regularisation term, in equation (12), does not influence the process a lot. Thus, FedProx and FedAvg are similar when the choice of μ is optimal and therefore in this particular example very small.



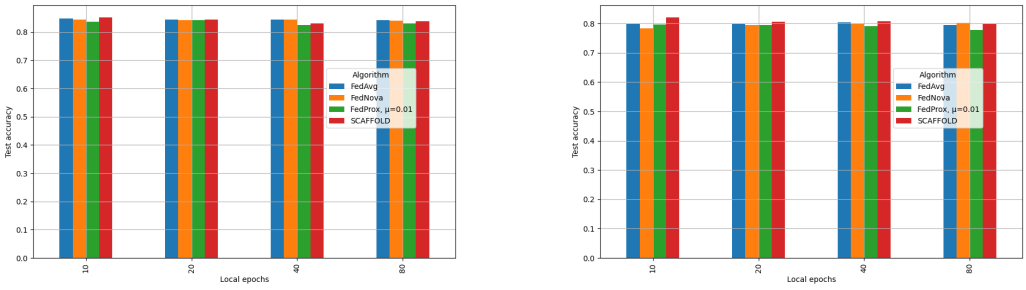
(a) IID

(b) $q_e \sim \text{Dir}(0.5)$

Figure 5.3: Test accuracy over rounds for the IID setting and the label-distribution imbalance setting

5.2.6 Robustness

By varying the number of local epochs E in the set $\{0, 20, 40, 80\}$ during the learning process we have plotted the final accuracy in figure 5.4, A.6 . These figures show that the number of local epochs has a significant influence on the accuracy of the algorithms. For example, when using the quantity of data imbalance or when using the label imbalance based on the Dirichlet distribution (figures 5.4b, A.6a, A.6b, A.6c), we first observe that the accuracy using the algorithms generally degrades as the number of local updates increases. This means that the algorithms are not robust to the number of local updates used in the process. Next, we observe that the accuracy for each of the algorithms depends on the local epochs and seems to be quite equal with little bit difference. For example Scaffold gives a high accuracy when the local epochs equals 10, 20 using label quantity imbalance with 2 labels per client but gives the lowest accuracy when the data quantity is imbalanced at local epoch=80.

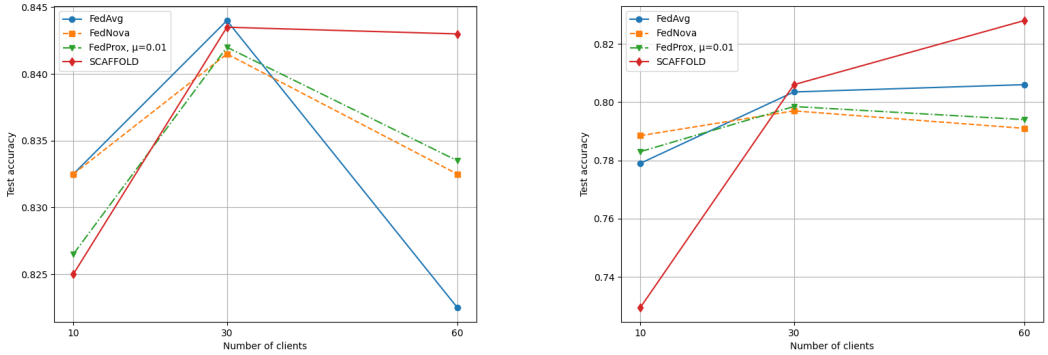


(a) IID (b) $q_e \sim \text{Dir}(0.5)$

Figure 5.4: Test accuracy over rounds, varying local epochs

5.2.7 Scalability

Here, we choose to varying the number of clients N to $\{10, 30, 60\}$ and reported the final accuracy given by the algorithms. The results are shown in figure 5.5a, where it can be seen that, in the IID data scenario, the accuracy increases significantly as the number of clients increases from 10 to 30 and then decreases from 30 to 60. The accuracy decreases using FedAvg is slightly high than when using Scaffold, FedNova and FedProx. In the case where the label distribution is imbalanced (figure 5.5b), using Scaffold always increases the accuracy while with the others, the accuracy generally increases when the number of clients increases from 10 to 30, but decreases slightly when the number increases from 30 to 60. In addition, using Scaffold in all the non-IID scenarios, always increases the accuracy. Moreover, using FedAvg and FedNova, the accuracy always decreases when the number of clients increases from 30 to 60. Furthermore, in the case where the label quantity is imbalanced (figures A.7a, A.7b), using FedProx decreases the accuracy when the number of clients increases from 10 to 30 and increases when the number of clients increases from 30 to 60.

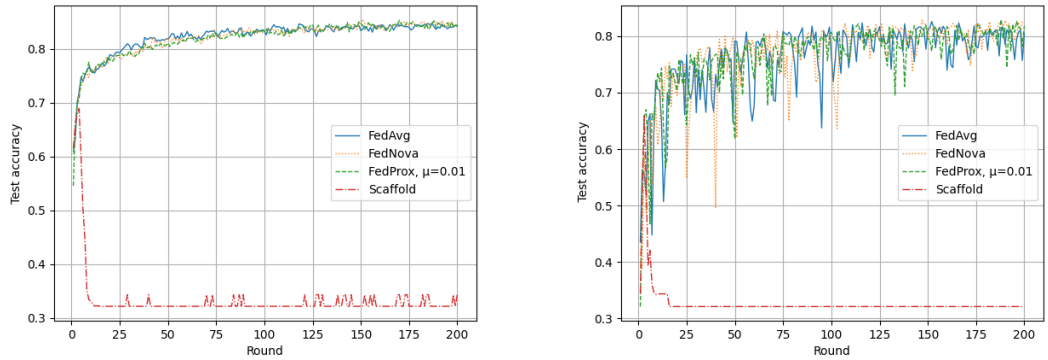


(a) IID (b) $q_e \sim \text{Dir}(0.5)$

Figure 5.5: Test accuracy over rounds, varying clients

5.2.8 Partial participation of clients in training

If we consider a large number of clients 80, but allow only 8 of them to participate in the learning process in the scenarios thus consider a fraction of $C = 0.1$, using a large number of communication rounds ($T = 200$), it can be seen in figure 5.6 that Scaffold has poor accuracy, whereas the others algorithms appear to be close. This is because, in Scaffold, the frequency of changes in the local control variate is low and in the other algorithms, only the gradients have different directions from one round to another.

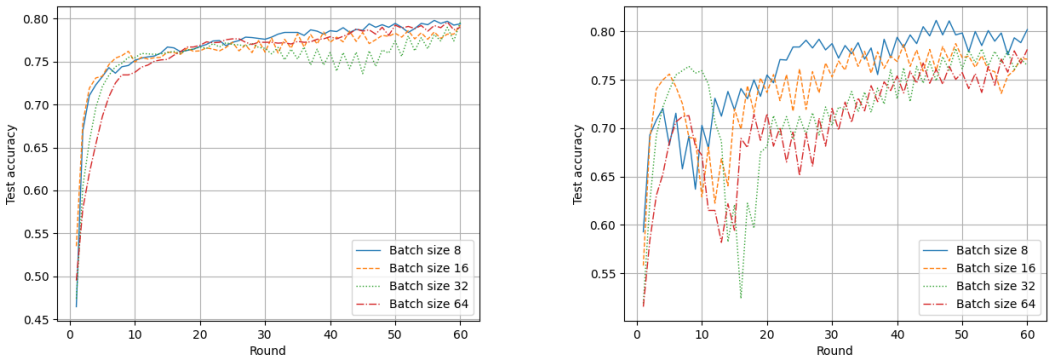


(a) IID (b) $q_e \sim \text{Dir}(0.5)$

Figure 5.6: Test accuracy over rounds, sampling client

5.2.9 Batch size, α and μ effects

Taking different values of batch sizes B in the set $\{8, 16, 32, 64\}$, figures 5.7, A.9 show that a large batch size slows down the learning process. More precisely, the relationship between batch size and the data using these algorithms is not very clear, but it may be representative. With FedNova, the accuracy is very unstable during the process, and we observe a significant decrease and increase from 20 to 30 communication rounds.



(a) FedAvg

(b) FedNova

Figure 5.7: Test accuracy over rounds, batch size changing

Moreover, now considering different values of μ in FedProx, figure 5.8 shows that a small value of μ (0.001) gives a high accuracy over 60 rounds and there is a small difference in the accuracy between using $\mu = 0.001$ and $\mu = 0.1$ at the end of the process. In addition, a high value of μ implies that a low accuracy is obtained during the learning.

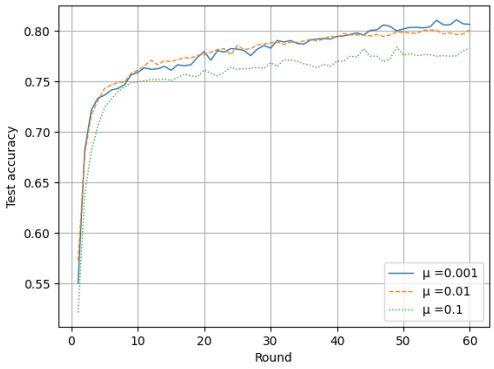
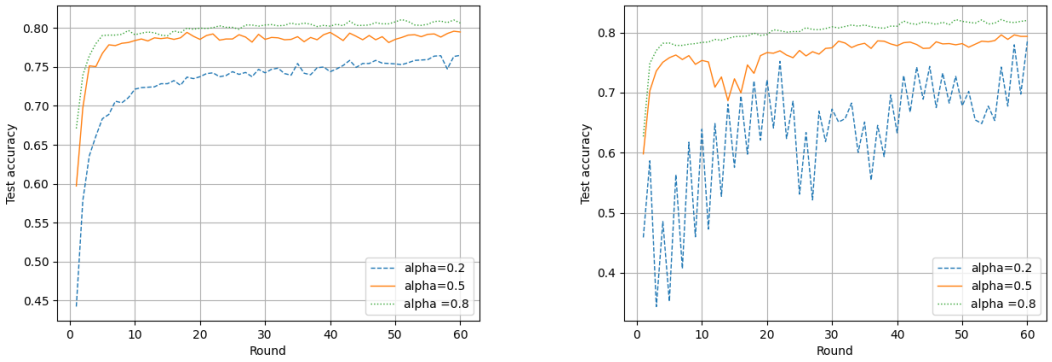


Figure 5.8: Test accuracy over rounds, μ parameter changes

Furthermore, by changing the α value in the set $\{0.2, 0.5, 0.8\}$ in the label distribution imbalance scenario using the different algorithms, it appears from figures 5.9, A.10 that the algorithms are highly impacted by α. The higher the α , the greater the accuracy. The accuracy using FedNova is very unstable when using a low $\alpha = 0.2$ than for the other algorithms. A value of $\alpha = 0.5$ and $\alpha = 0.8$ appears to give close accuracy after 60 rounds of communication.



(a) FedAvg (b) FedNova

Figure 5.9: Test accuracy over rounds, label imbalance, α varying

5.3 Simulated data

5.3.1 Dataset

In this part, in order to evaluate the performance of the algorithms under different scenarios, we generate different datasets and solve a classification problem where the target variable has 3 classes using make_classification[3] method form scikit-learn. We vary the number of features $d \in \{15, 50\}$ and the number of samples $n_k \in \{10000, 20000\}$, we fixed the number of local epochs to $E = 10$, the number of communication rounds to $T = 60$ and the number of devices to $N = 30$.

The matrix of observations X is generated of independent observations as:

$X = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ a_{21} & a_{22} & \dots & a_{2d} \\ \dots & \dots & \dots & \dots \\ a_{n_k1} & a_{n_k2} & \dots & a_{n_kd} \end{pmatrix}$, such that $\forall a_{ij} \sim \mathcal{N}(0, 1)$, where $i \in \{1, 2, ...n_k\}, j \in \{1, 2, ..., d\}$.

This can be seen in the figure below for a distribution of a feature with the number of samples $n_k = 10000$:

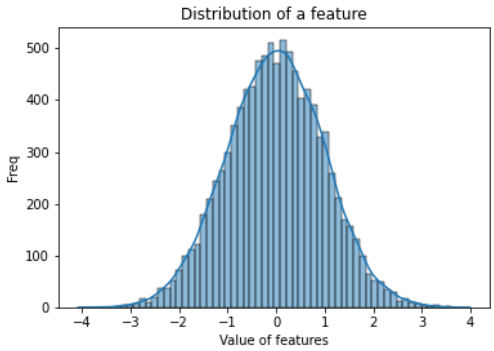


Figure 5.10: Distribution of a feature

The target variable Y is generated in the sense that :

$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_{n_k} \end{pmatrix} = \arg\max_{cl} \left(\mathbf{w}_{cl} \cdot X + \epsilon_{cl} \right)$, where X denotes the features created above, cl the class, ϵ_{cl} the noise added in the algorithm by the parameter of the function $flip_y = 0.01$ which specified that 1% of the labels are randomly flipped, and $\mathbf{w}_{cl}$ the random coefficient of class cl generated in the function.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html`

We evaluate the methods based on accuracy, and weighted F1_score. The weighted F1_score [4] is defined as : weighted F1_score $= \sum_{c=1}^{C} l\mathrm{w_c} \times$ F1_score, with Cl the number of class labels in the target variable, $\mathrm{w_c}$ refers to the proportion of class support relative c to the sum of all supports values Cl, and

F1_score $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$, where precision is the number of true positives results divided by the number of all samples predicted as positives and recall is the number of true positives results divided by the number of all samples that should have been identified as positives. Next, we consider the same scenarios as in section 5.2.1 except the one with data distribution imbalance ($q \sim dir(\alpha)$).

5.3.2 Results

For each design, we generated the different datasets 100 times. A total of 12 designs have been generated and the results obtained are presented in Table 5.1. For all the scenarios, we observe first that in the setting IID , we obtained the best accuracy compared to non_IID setting. This make sense since in the IID setting, all the client's data have the same distribution and each sample is independent of the others. This is not the case in the non_IID setting where the fact that the data are distributed to clients in different ways make the global training model difficult. Also, in federated learning, since local models are aggregated on the server to make a global model that generalises well the participating devices, they are more similar in the IID setting, hence the global model there is more effective.

On the other hand, in the non_IID setting, the label imbalance based on the Dirichlet distribution leads to worse accuracy than label imbalance based on a quantity. This is explained by the fact that in the label imbalance based on quantity, it's only the quantities of classes which are varying whereas in the case of label imbalance based on distribution, some devices have a high concentration of certain class labels while others have few of those class labels, thus the trained model is not a good representative of all participating devices.

As the number of features increases, we observe that the accuracy becomes smaller. This may be due to the fact that with a high number of features, the model has difficulty to identify relevant relationships in the data and as the feature space becomes sparser, it is more difficult for the model to generalise to unseen data. It can also occur by the fact that with a large number of features, the model is too complex, and if the number of observations is too small, this increases the risk of overfitting, i.e. the fit on the training data is too tight and then, on the test data, the accuracy may be lower.

We observe also that the accuracy is higher when the number of observations is

[4] https://www.kdnuggets.com/2023/01/micro-macro-weighted-averages-f1-score-clearly-explained.html

high. As the number of observations increase, the model has much more data to fit on, which reduces the risk of overfitting, and allows it to be generalised to unseen data by capturing a variety of patterns. Next, in all the scenarios, accuracy is very close to weighted F1_score meaning that the model may consistently perform well or that the precision and recall for each class are high or low during the process.

Finally, in all the designs, Scaffold consistently has the best accuracy compared to the other algorithms. Since it uses control variates, it will try to reduce the variance of the stochastic gradients, accelerating the convergence of the global model and improving the accuracy. Therefore in the non_IID setting, when the data distribution is not the same across the devices, Scaffold can achieve better performance because it mitigates the problem of heterogeneity. However, it has close standard deviation to the others in all the designs. Thus, it seems that the algorithms are not very different in terms of these metrics.

Table 5.1: Average accuracy, F1_score for differents designs on 100 simulations

	design	FedAvg				FedProx				Scaffold				FedNova			
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
		acc	acc	F1	F1	acc	acc	F1	F1	acc	acc	F1	F1	acc	acc	F1	F1
$n_k = 10000$ $d = 15$	IID	.874	.020	.874	0.021	.867	.020	.869	0.020	.887	.020	.887	.020	.869	.020	.867	.021
	$p_k \sim \text{Dir}(0.5)$	.829	.040	.828	.043	.836	.025	.836	.025	.857	.038	.856	.043	.824	.037	.822	.041
	#lab=2	.854	.024	.854	.024	.859	.026	.859	.026	.879	.029	.878	.029	.856	.025	.856	.026
	IID	.819	.021	.819	.021	.816	.020	.816	.020	.823	.023	.821	.023	.813	.022	.812	.022
$p_k = q_e \sim \text{Dir}(0.5)$		.766	.036	.8764	.041	.777	.035	.776	.038	.789	.037	.788	.042	.757	.035	.753	.041
	#lab=2	.792	.024	.792	.024	.804	.027	.804	.027	.822	.027	.821	.027	.799	.024	.798	.025
$n_k = 20000$ $d = 15$	IID	.893	.017	.892	.017	.893	.017	.893	.017	.904	.016	.904	.016	.892	.017	.892	.017
	$p_k \sim \text{Dir}(0.5)$	.865	.035	.864	.039	.859	.034	.858	.036	.890	.034	.889	.037	.880	.029	.880	.031
	#lab=2	.876	.024	.876	.024	.879	.025	.879	.025	.901	.025	.901	.025	.883	.024	.882	.024
$p_k = q_e \sim \text{Dir}(0.5)$	IID	.852	.020	.851	.020	.851	.020	.851	.020	.860	.018	.859	.018	.851	.019	.851	.019
		.807	.033	.806	.036	.815	.032	.815	.035	.833	.035	.832	.039	.815	.026	.815	.028
	#lab=2	.824	.023	.824	.023	.832	.027	.832	.027	.848	.026	.847	.026	.832	.025	.832	.025

5.4 Real data

5.4.1 Dataset

In this section, the data we have used is related to strokes,and it contains 10 features with above 5000 observations. The dataset is used to predict whether a patient is likely to get stoke or not. Therefore response variable is *stroke* and the others inputs parameters are age, smoking status, hypertension, ... This dataset is an educational dataset which can be find in *Kaggle*[5] .

Before any analysis, some preprocessing steps are necessary. First, we make individual plots of features and we decide to remove missing values presents in the *body_mass_index (bmi)* variable by replacing them with the mean and encoded categorical features to dummies variables. The figure below presents the distribution of classes in the target variable. It shows that the dataset has more observations with class 0 than with the class 1.

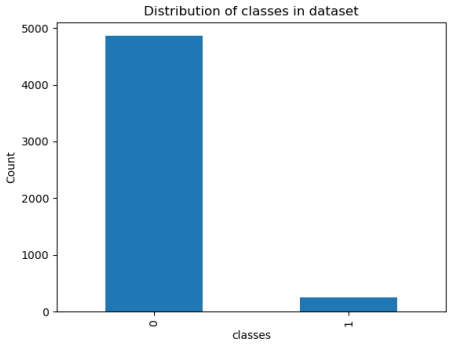


Figure 5.11: Distribution of classes in target variable on real data

Four our experiments, we choose $N = 10$ clients, the number of locals epochs $E = 20$, the number of rounds $T = 40$, batch size $B = 32$ and a learning rate $\eta_s = 0.00001$. To evaluate the algorithms, we use the same scenarios as in the 5.2.1 section. Then, we consider two models in experiments. The first one, Logistic regression, estimates the probability that an event occurs and the second, Deep Neural Network (DeepNN), is a neural network with multiples layers (in this case 3 with 128, 64, 32 neurons respectively). In addition, we choose to add an optimizer AdamW [6] with the aim of decoupling the weight decay and loss based on gradient updates instead of combining weight decay and gradient updating as in Adam.

[5] `https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset`
[6] `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`

Next, since our dataset has two classes for the target variable, we will not only use the accuracy as metric but also the binary cross_entropy loss function defined as: $\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$ with $\hat{y}_i = \frac{1}{1+e^{-(\gamma_0+\gamma_1 x_i)}}$ as in the section 3.2.2 for logistic regression and for DeepNN,
$\hat{y}_i = \sigma^{(La)}\left(\mathbf{W}^{(La)}\sigma^{(L-1)}\left(\mathbf{W}^{(La-1)}\cdots\sigma^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)}\right)\cdots + \mathbf{b}^{(La-1)}\right) + \mathbf{b}^{(La)}\right)$
where

- $\hat{y}_i$: The predicted value for the i-th sample, corresponding to the actual target y_i.
- $\mathbf{x}_i$: The input feature vector for the i-th sample.
- $\mathbf{W}^{(la)}$ and $\mathbf{b}^{(la)}$: The weight matrix and bias vector for layer la.
- $\sigma^{(la)}(\cdot)$: The activation function applied at layer la.
- La : The total number of layers in the network.

Thereafter, we set up two setups; the first one when the classes in the target variable are consider imbalanced in the dataset before splitting called *target imbalance* and the other called *target balance* when the classes in the target variable are consider balanced. For that, we weighted each class of the target variable in the training model and we use the binary cross_entropy_logit with the weight defined for the purpose of calculating the loss, taking into account the distribution of each class. The main difference here with the other setting is that, we make our set up in the dataset before splitting it and then, we use the same settings as in section 5.2.1 to share the data between clients.

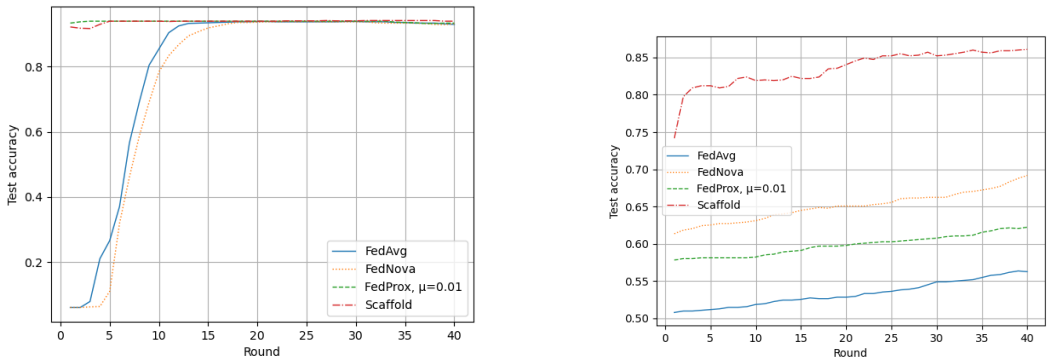
5.4.2 Results

During the process, we made a number of observations based on the two measures chosen for the evaluation.

Accuracy metric results

In this part, by visualising only the accuracy plots, it seems first that over the process, in most situations, Scaffold has a higher accuracy than the others using Logistic regression. Next, using again Logistic regression, the algorithms (except Scaffold) don't reach similar accuracy after performing the same number of rounds communication than when using a DeepNN model. Meaning that using a DeepNN of the model seems to converge faster and more robustly the accuracy in the scenarios than using a Logistic regression model which may require more rounds to achieve similar accuracy. This might be so given that DeepNN has the ability to capture complex non linear relationships in data.

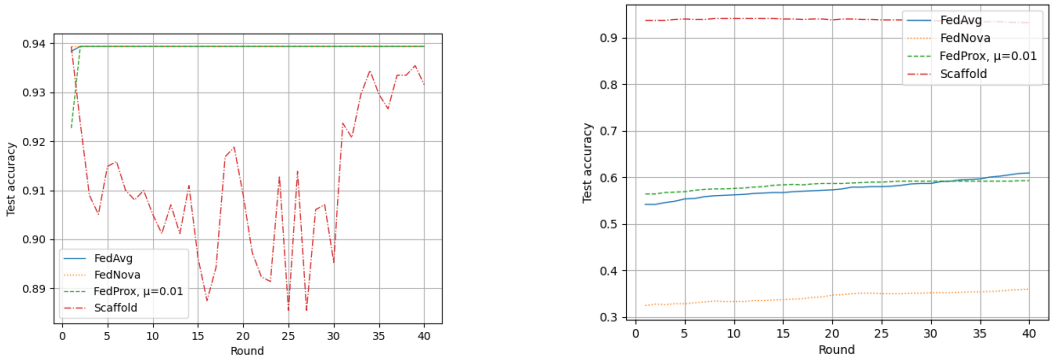
On the other hand, considering the fact that the data are balanced in term of class labels in the target variable before sharing them between the clients (target balance), we see in figures A.13, A.12, A.11, 5.12 that, in most of the case, the algorithms quickly converge to a stable accuracy. Despite the fact that Scaffold also converges and maintain high accuracy in the DeepNN model, it sometimes encounters considerable difficulties (figures A.18, A.13). Furthermore, using the Logistic regression model, we see that the algorithms FedAvg, FedProx, FedNova do not reach similar accuracy as when using DeepNN. In addition, especially, in the data distribution imbalance ($q \sim dir(\alpha)$) scenario, figures A.13, the algorithms improve their accuracy but not as effectively like in DeepNN.



(a) DeepNN (b) Logistic Regression

Figure 5.12: Accuracy in IID target balance setup

Looking at figures A.16, A.15, A.14, 5.13 where we are in case of target imbalance, FedProx requires more rounds(30) to achieve similar accuracy as FedAvg or FedNova in an IID scenario, and as the number of rounds becomes large, the accuracy using Scaffold seems to become unstable using DeepNN model. Also, as in the case of class labels balance, regardless of the model used, Scaffold maintains higher accuracy than the other algorithms. Furthermore, FedAvg seems to have the lowest accuracy using Logistic regression except in the IID scenario where compared to the others scenarios, it has an improvement of the accuracy.



(a) DeepNN (b) Logistic Regression

Figure 5.13: Accuracy in IID target imbalance setup

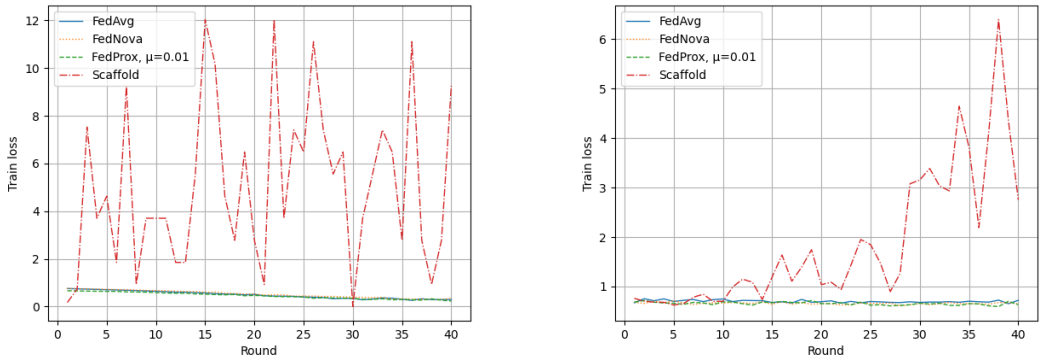
Generally, we see that Scaffold shows high accuracy across both DeepNN and Logistic regression models relative to the others algorithms. The accuracy of the algorithms is higher when we used DeepNN and smaller when we used Logistic regression models. Having target balance or imbalance has an impact on the performance and may be taking into account when using the algorithms. Before any partial conclusion on whether Scaffold is the algorithm to choose for this dataset, it is important not to consider the accuracy alone. In the following, we will analyse the behaviour of the loss during the learning process.

Loss metric results

Concerning the training loss in figures 5.14 and A.18, A.19, A.17, A.20, A.22, A.23, A.21, it seems that Scaffold presents a huge instability during the learning process. This could be due to various factors, such as the fact that the hyper-parameters chosen may not be optimal for this algorithm, particularly the learning rate. Also, the cause may be due to the fact that the variance in gradient updates is taken into account in Scaffold as the data are decentralised, with each client computing local gradient updates on its own subset of data before sending them in to the central server. And as their data in the different scenarios are very different, the variance will be consequently very large. Therefore, the control variates used in order to correct client drift and ensure that the global model does not deviate too far from the local updates, are not optimal for this data case.

In addition, the fact that the loss in the other algorithms decreases during the process could mean that certain aspects of the data are not taken into account by these algorithms, whereas Scaffold manages to capture them such as the variability in the data.

Furthermore, another factor could be a potential overfitting. The model may have memorised patterns containing in the training set instead of learning to generalise them, which is why the test accuracy is high and yet the training loss very unstable. Moreover, the instability in the training loss of the Scaffold algorithm can come from the fact that during the learning process, the model eventually finds a good minima which works well on the training data.

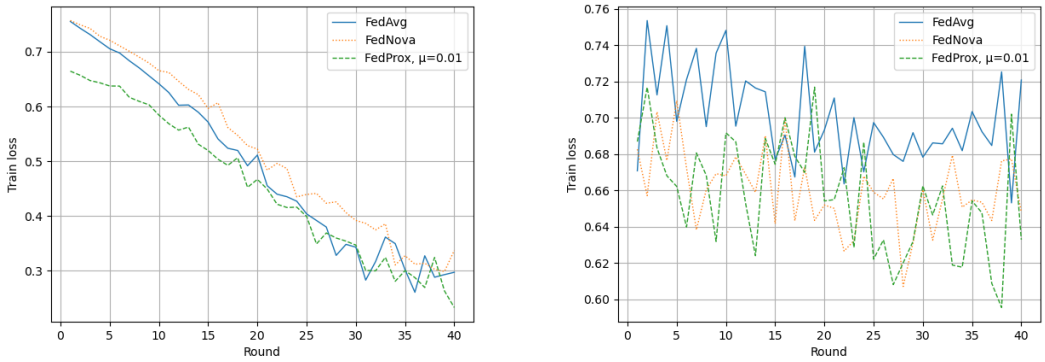


(a) DeepNN (b) Logistic Regression

Figure 5.14: Loss in IID target balance setup

Exploring training loss of FedAvg, FedProx, FedNova algorithms to see if the instability of the training loss of Scaffold is due to the fact that it takes into account certain aspects than the others or if the instability is really due to the hyperparameters chosen, we visualise figures A.27, A.29, A.30, A.28, 5.15, A.25, A.26, A.24 where we choose to remove the scaffold algorithm in order to see if effectively the training loss decrease over the round in the different scenarios. These figures confirm that the choice of our hyperparameters especially learning rate have a high impact in the training loss especially when we are in case of label imbalance (based on distribution or on quantity) using DeepNN model. Using Logistic regression, we see that in all the scenarios and setups, the are a lot of fluctuations in the training process which means that the model has difficulty to update its weights and is very unstable.

In this way, considering $\eta_s = 0.00001$ is not a good choice for this data under these scenarios and algorithms. Therefore we can not make a choice between Scaffold, FedAvg, FedProx or FedNova as a suitable algorithm for this dataset.



(a) DeepNN (b) Logistic Regression

Figure 5.15: Loss in IID target balance setup without Scaffold

Conclusion

In automation and artificial intelligence, Federate Learning is a plus and a boost, that helps to solve issues around private and amount of available data. This dissertation is a journey through the concept of Federate Learning, by discussing the challenges that come from and proposing the solutions that can help to perform it. We begin with a brief history and talking about parts and types of Federated learning.

Further on, a presentation of some Federated learning algorithms in the literature, after a mathematical formulation of the Federated learning objective and a discussion about the properties of the algorithms allows us to understand how to implement these algorithms.

We tune then the hyperparmeters on simulated data. The results obtained underline the fact that with some specific hyperparameters, effective communication, scalability, robustness can be done and high performance of the models is maintained. We recommend to choose the right hyperparameters depending on the data available. When having a considerable amount of data, our analysis suggests to consider μ large, η_s, α and the batch size very small.

A simulation process on data, shows that algorithms FedAvg, FedProx, FedNova and Scaffold can handle effectively heterogeneity of data in divers scenarios and that they can be very close in terms of accuracy, and weighted F1_score. Considering the label imbalance based on quantity (#lab=1), when having more than two labels, is an extreme case since the clients has no diversity in the labels and our analysis shows that the model will not perform well. Also, the number of participating client devices and the model chosen for implementation are important. A model capable of handling non-linear relationships may be preferred in situations where we have a large amount of data and the relationships in it are not linear, but we need to be careful about the cost of complexity.

In this study, it is important to notice that the choice of a federated learning algorithm is not only based on the precision but also on the loss during the training model. This is to avoid considering a model based on a local minimum and unstable.

To improve this work, we suggest other areas of research, such as the search for algorithms that allow hyperparameters to be adjusted efficiently, quickly and

securely for their own use in models and according to the data available. Although each algorithm has been designed to respond to a specific problem, what set of data or scenario would be the most conducive to a coherent visual comparison of federated learning algorithms? How to make a real collaborative learning in order to improve research in health domain are just a few of potential questions for further research.

Appendix A

Results

A.1 Representation scenarios

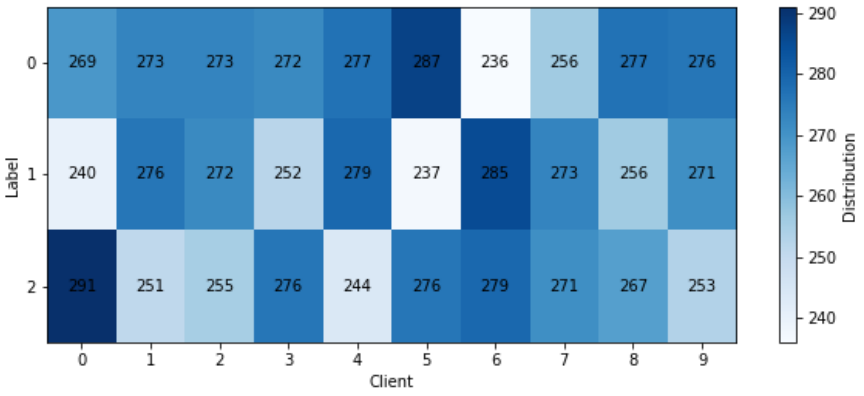


Figure A.1: Label distribution for the IID setting in the learning set

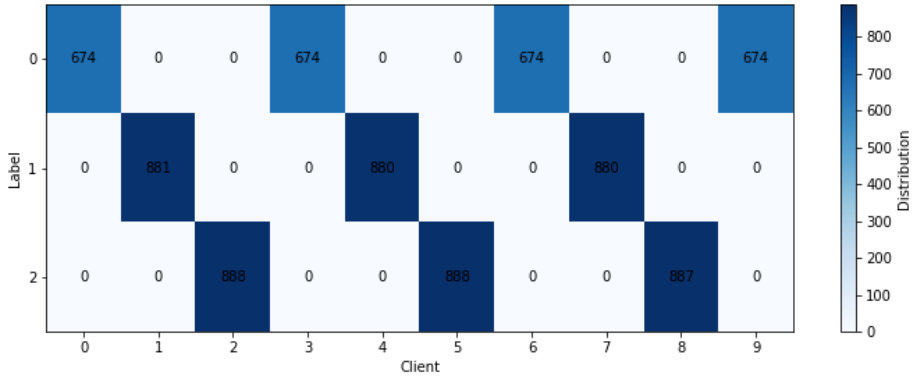


Figure A.2: Quantity-based label imbalance, label_per_client=1

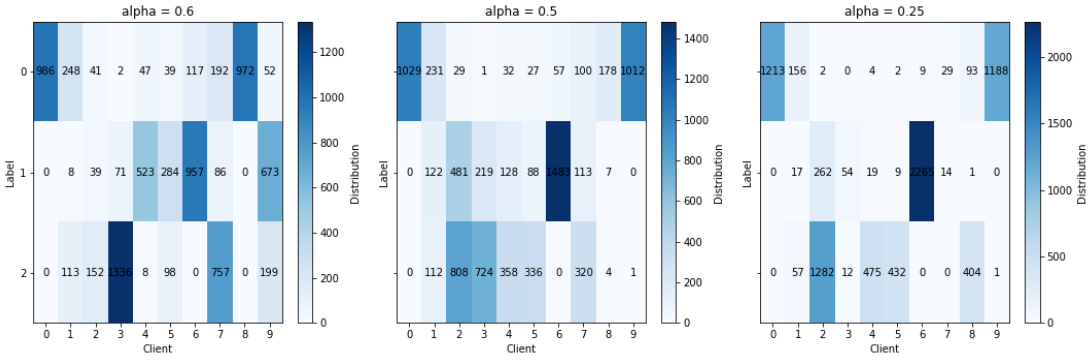
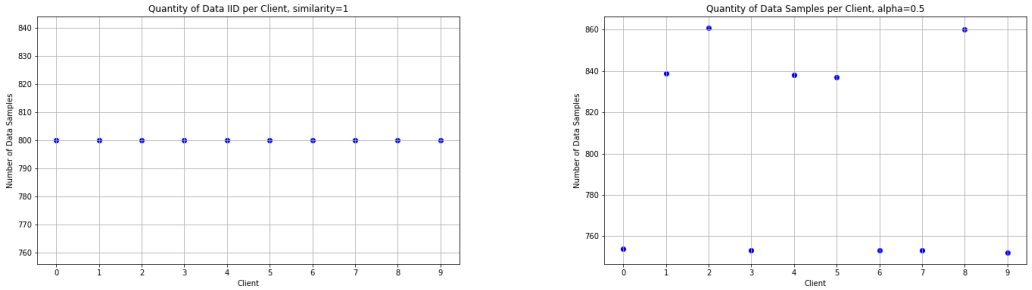


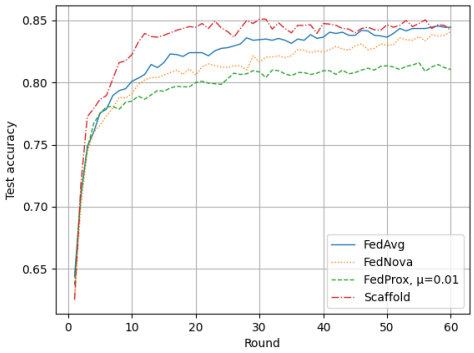
Figure A.3: Distribution-based label imbalance



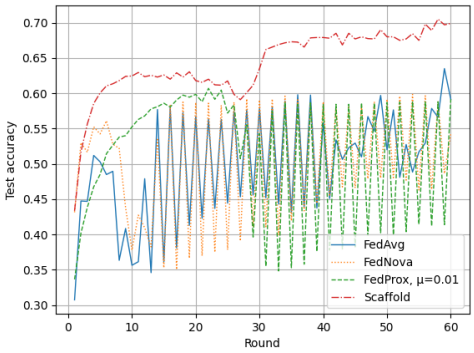
(a) IID distribution (b) Quantity of data imbalance

Figure A.4: Scenarios IID and Quantity of data imbalance

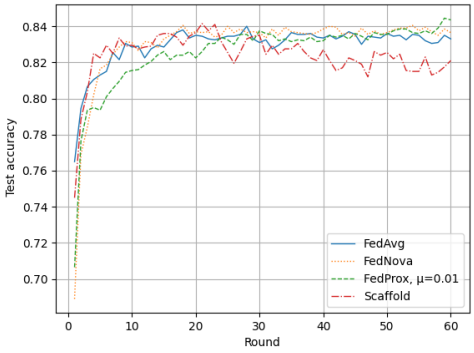
A.2 Test Accuracy over rounds



(a) 2 labels per client



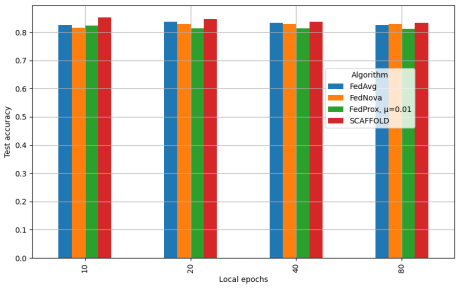
(b) 1 label per client



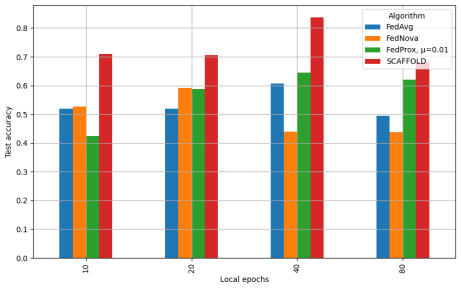
(c) $q \sim \text{Dir}(0.5)$

Figure A.5: Test accuracy over rounds of label-quantity imbalance, quantity imbalance

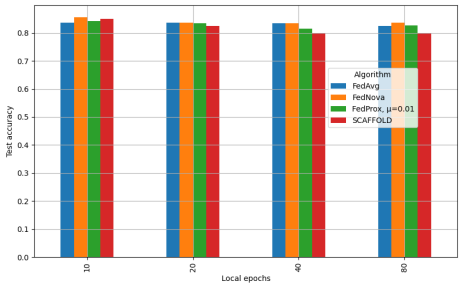
A.3 Test accuracy over rounds, varying epochs



(a) 2 labels per client



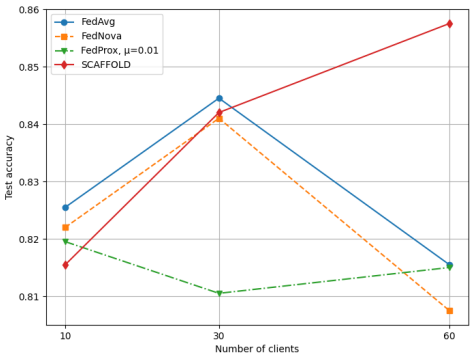
(b) 1 label per client



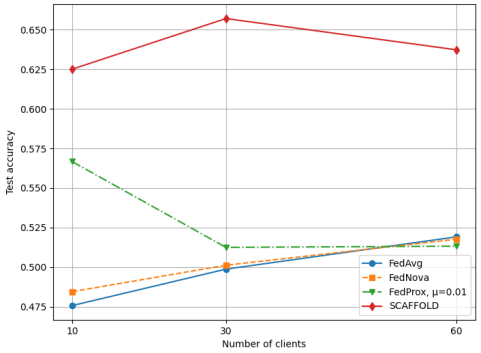
(c) $q \sim \text{Dir}(0.5)$

Figure A.6: Test accuracy over rounds, varying local epochs

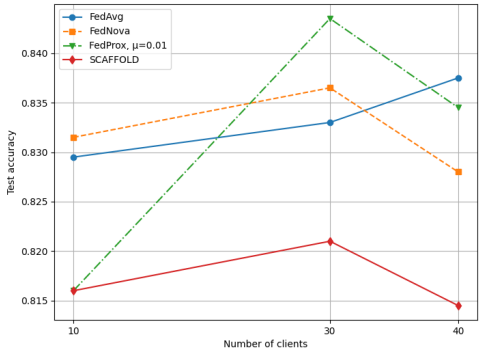
A.4 Test accuracy over rounds, varying the number of clients



(a) 2 labels per client



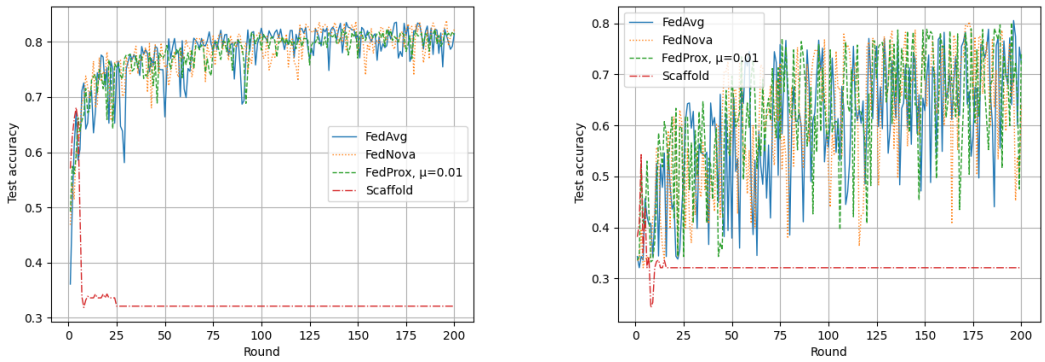
(b) 1 label per client



(c) $q \sim \text{Dir}(0.5)$

Figure A.7: Test accuracy over rounds, varying clients

A.5 Test accuracy over rounds, partial participation

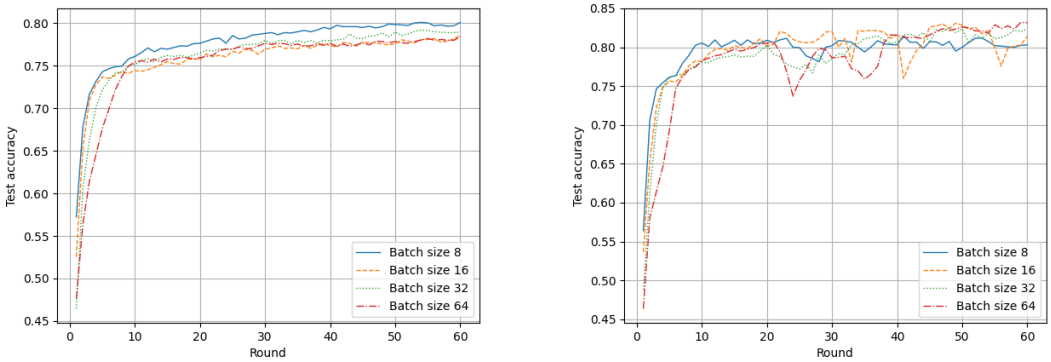


(a) 2 labels per client

(b) 1 label per client

Figure A.8: Test accuracy over rounds, sampling client

A.6 Test accuracy over rounds, varying batch size

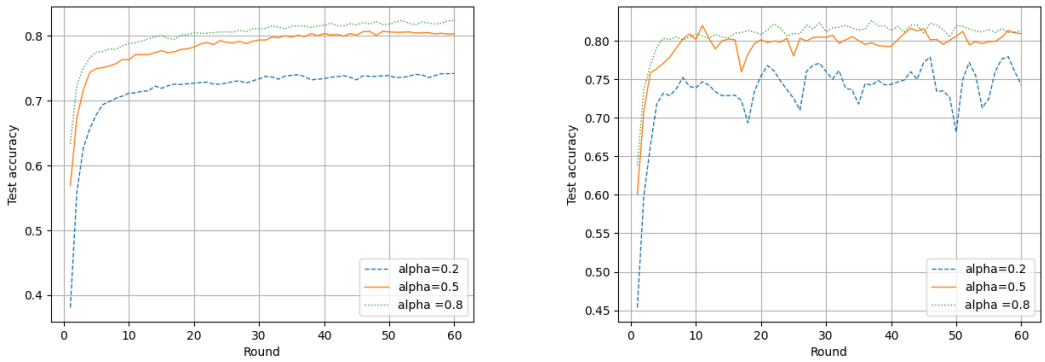


(a) FedProx

(b) Scaffold

Figure A.9: Test accuracy over rounds, Batch size changing

A.7 Test accuracy over rounds, varying α

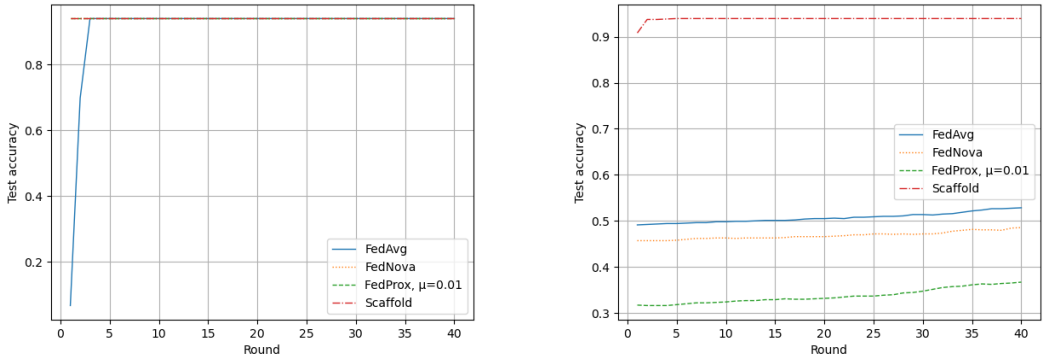


(a) FedProx (b) Scaffold

Figure A.10: Test accuracy over rounds, label imbalance, α changing

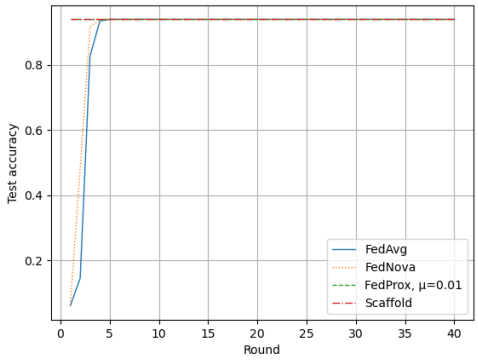
A.8 Real data

A.8.1 Accuracy metric results

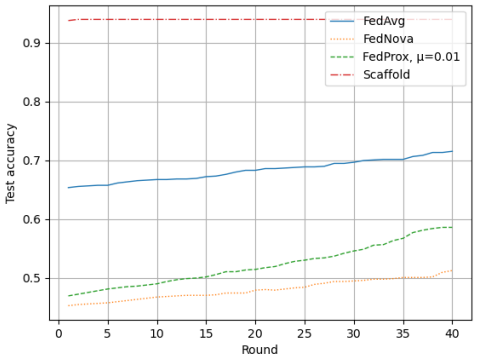


(a) DeepNN (b) Logistic Regression

Figure A.11: Accuracy in $q_e \sim dir(\alpha)$ target balance setup

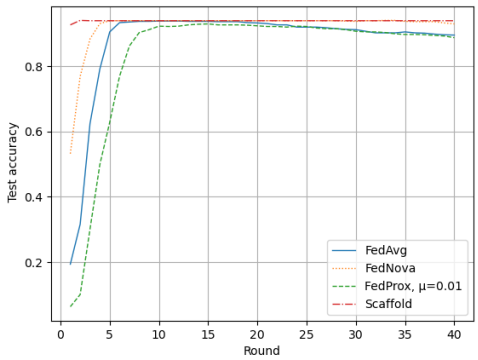


(a) DeepNN

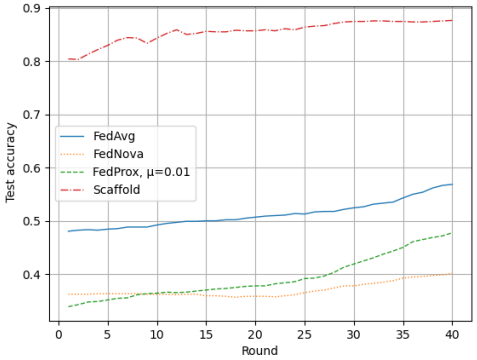


(b) Logistic Regression

Figure A.12: Accuracy in $\#lab = 1$ target balance setup

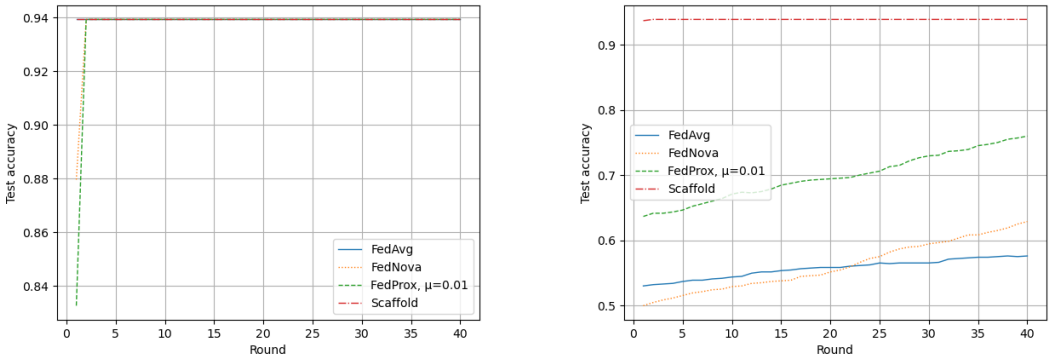


(a) DeepNN



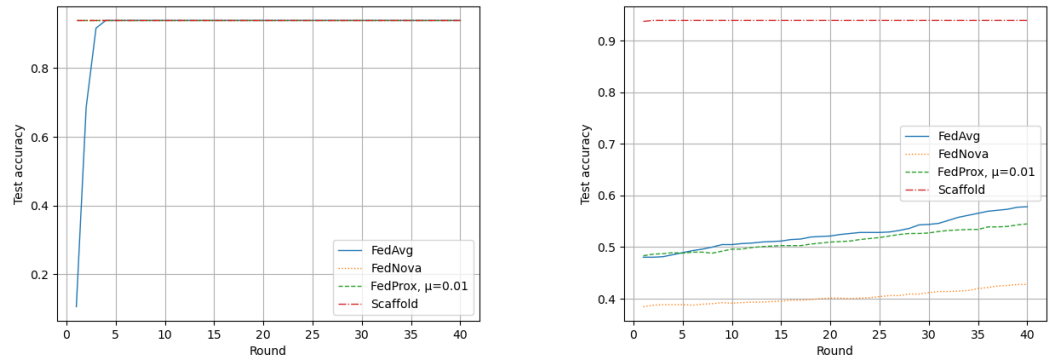
(b) Logistic Regression

Figure A.13: Accuracy in $q \sim dir(\alpha)$ target balance setup



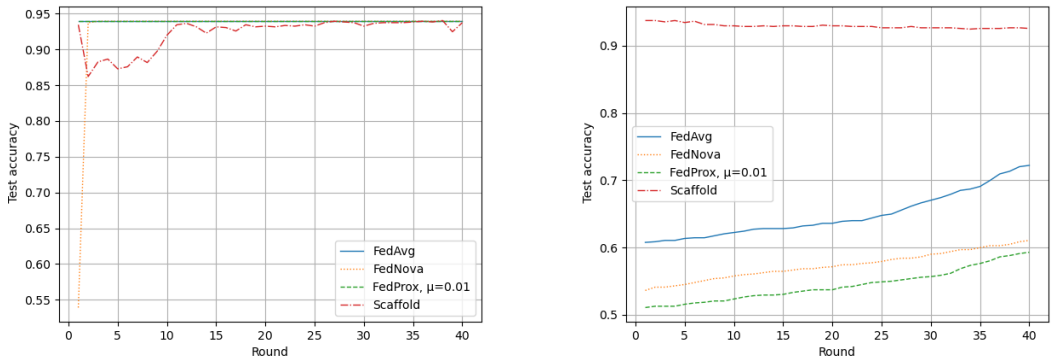
(a) DeepNN (b) Logistic Regression

Figure A.14: Accuracy in $q_e \sim dir(\alpha)$ target imbalance setup



(a) DeepNN (b) Logistic Regression

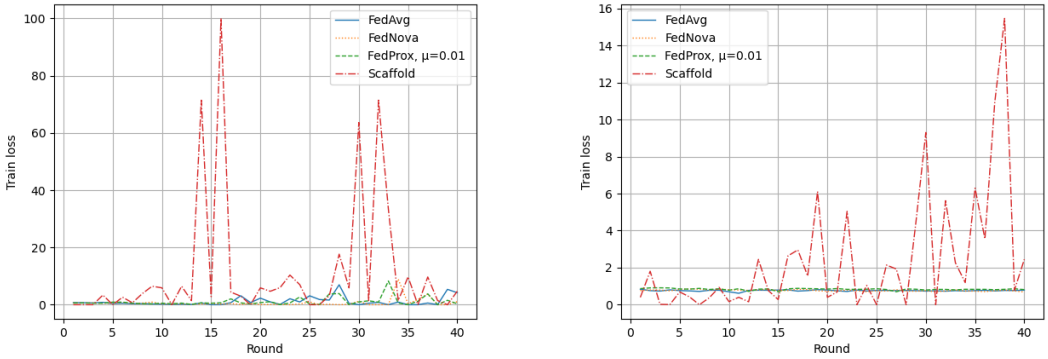
Figure A.15: Accuracy in $\#lab = 1$ target imbalance setup



(a) DeepNN (b) Logistic Regression

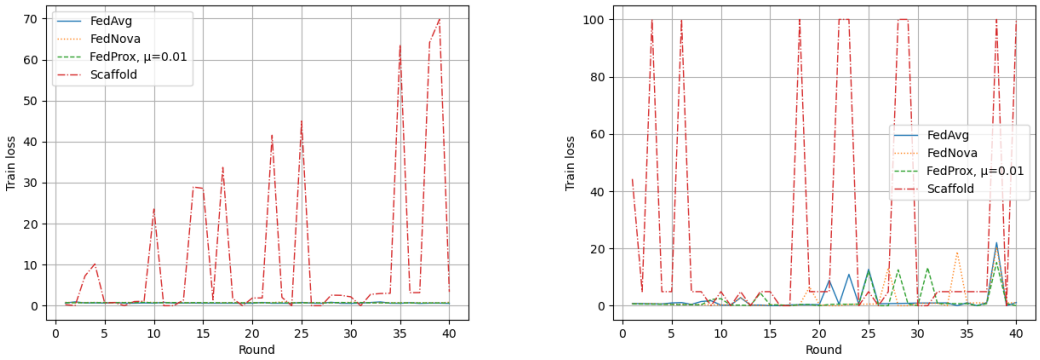
Figure A.16: Accuracy in $q \sim dir(\alpha)$ target imbalance setup

A.8.2 Loss metric results



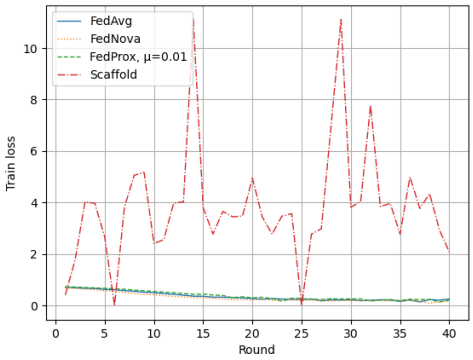
(a) DeepNN (b) Logistic Regression

Figure A.17: Loss in $q_e \sim dir(\alpha)$ target balance setup

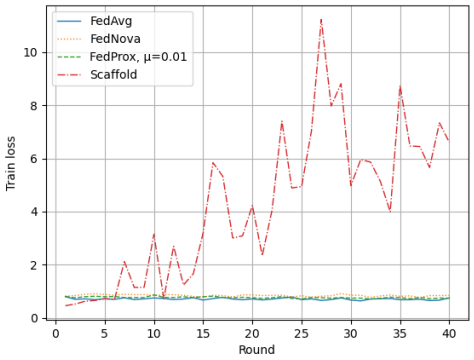


(a) DeepNN (b) Logistic Regression

Figure A.18: Loss in $\#lab = 1$ target balance setup

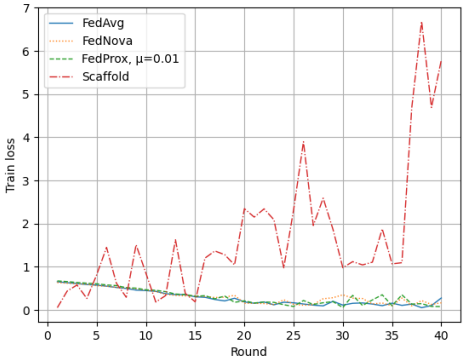


(a) DeepNN

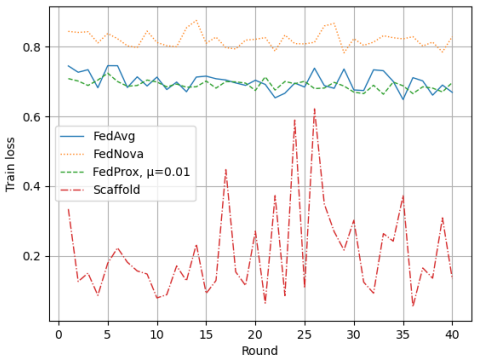


(b) Logistic Regression

Figure A.19: Loss in $q \sim dir(\alpha)$ target balance setup

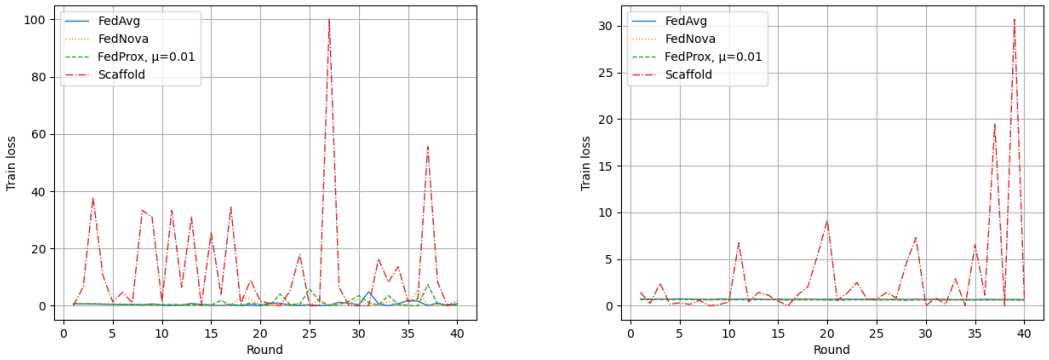


(a) DeepNN



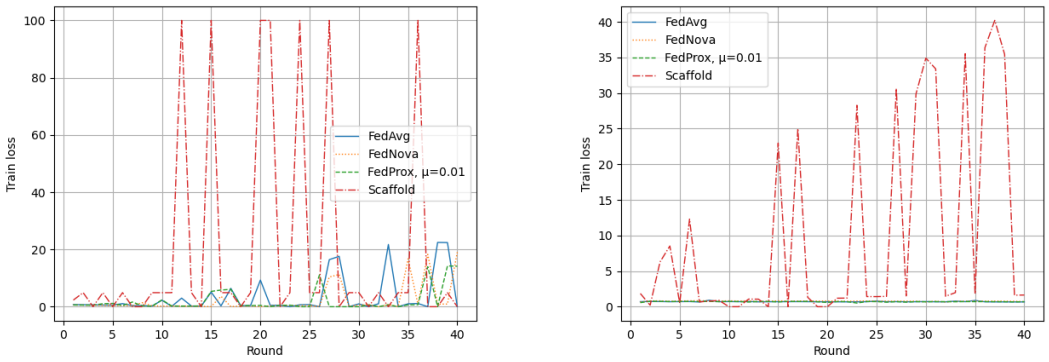
(b) Logistic Regression

Figure A.20: Loss in IID target imbalance setup



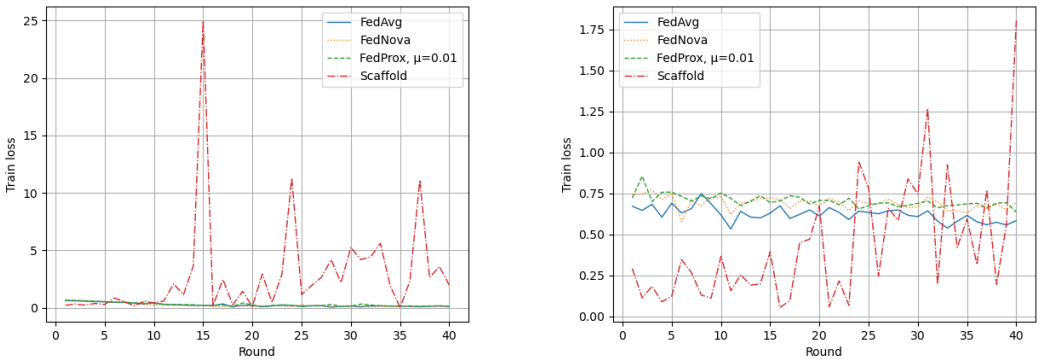
(a) DeepNN (b) Logistic Regression

Figure A.21: Loss in $q_e \sim dir(\alpha)$ target imbalance setup



(a) DeepNN (b) Logistic Regression

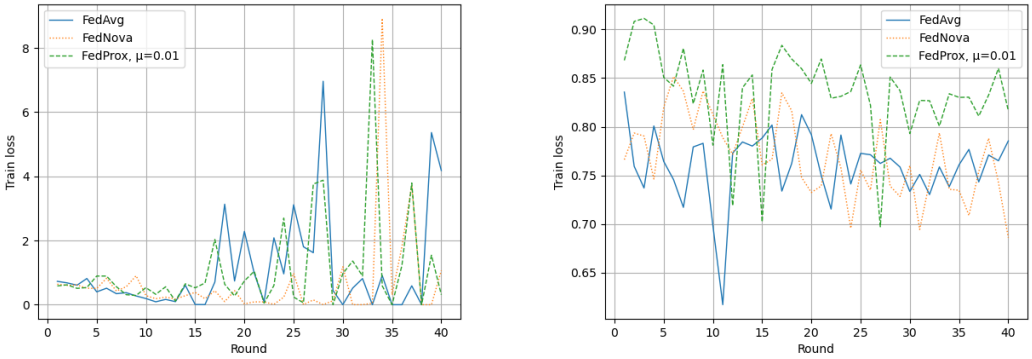
Figure A.22: Loss in $\#lab = 1$ target imbalance setup



(a) DeepNN (b) Logistic Regression

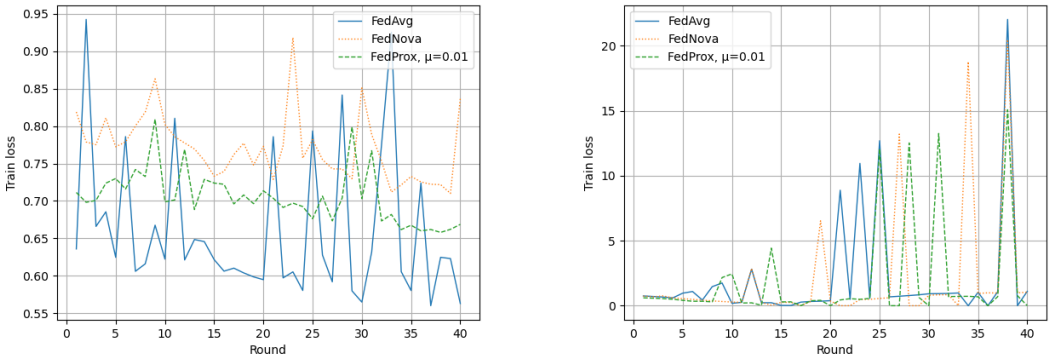
Figure A.23: Loss in $q \sim dir(\alpha)$ target imbalance setup

A.8.3 Loss metric results without scaffold



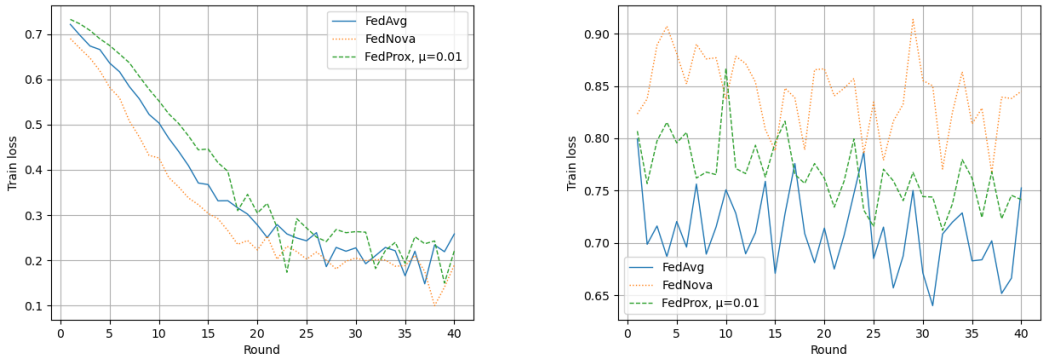
(a) DeepNN (b) Logistic Regression

Figure A.24: Loss in $q_e \sim dir(\alpha)$ target balance setup without Scaffold



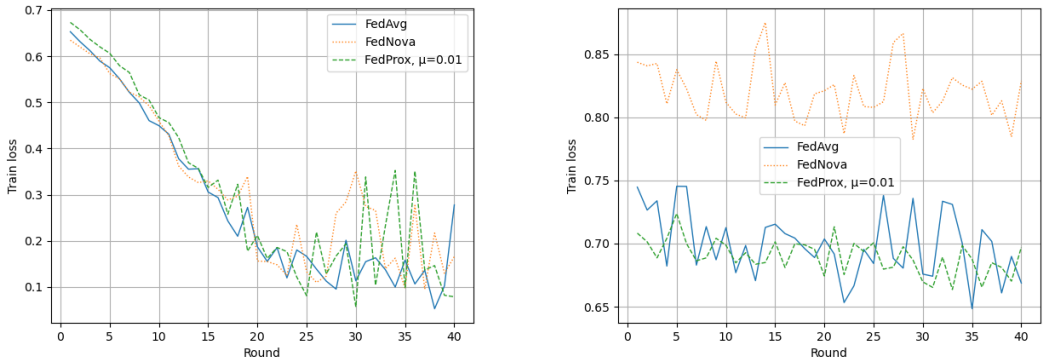
(a) DeepNN (b) Logistic Regression

Figure A.25: Loss in $\#lab = 1$ target balance setup without Scaffold



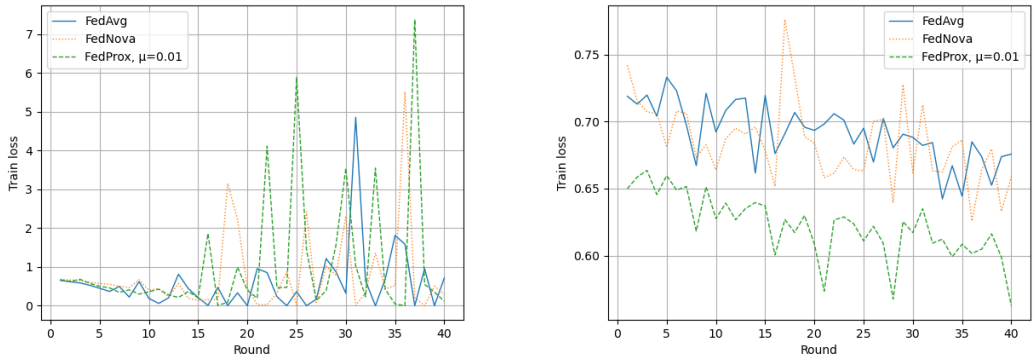
(a) DeepNN (b) Logistic Regression

Figure A.26: Loss in $q \sim dir(\alpha)$ target balance setup without Scaffold



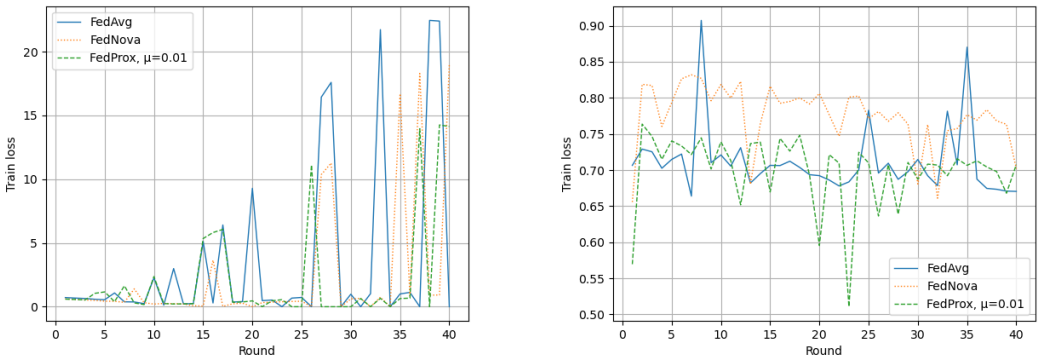
(a) DeepNN (b) Logistic Regression

Figure A.27: Loss in IID target imbalance scenario without Scaffold



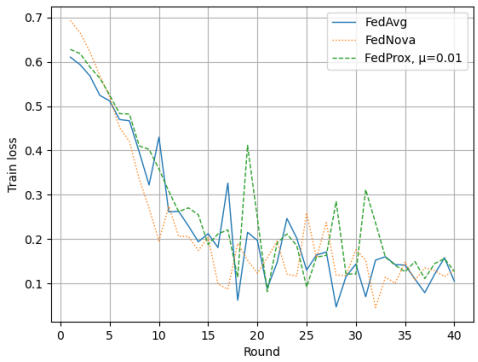
(a) DeepNN (b) Logistic Regression

Figure A.28: Loss in $q_e \sim dir(\alpha)$ target imbalance setup without Scaffold

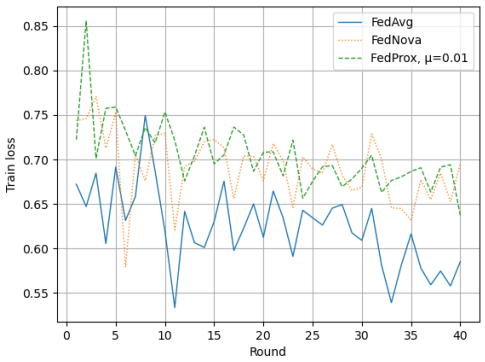


(a) DeepNN (b) Logistic Regression

Figure A.29: Loss in $\#lab = 1$ target imbalance setup without Scaffold



(a) DeepNN



(b) Logistic Regression

Figure A.30: Loss in $q \sim dir(\alpha)$ target imbalance setup without Scaffold

Bibliography

Azim Akhtarshenasa, Mohammad Ali Vahedifara, Navid Ayoobib, Behrouz Mahamc, and Tohid Alizadehc. Federated learning: A cutting-edge survey of the latest advancements and applications. arXiv:2310.05269v1, 2023.

Desai Rahul Babasaheb and Indhumathi Raman. Survey on fault tolerance and security in mobile ad hoc networks (manets). *2018 3rd International Conference for Convergence in Technology (I2CT)*, pages 1–5, 2018.

Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. arXiv:1812.00984v2, 2019.

Kallista Bonawitz, Peter Kairouz, Brendan Mcmahan, and Daniel Ramage. Federated learning and privacy. *Communications of the ACM*, 65(4):90–97, 2022.

Junyi Chai and Xiaoqian Wang. To be robust and to be fair: Aligning fairness with robustness. arXiv:2304.00061, 2023.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2022.

Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. *Financial Cryptography and Data Security 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*, pages 143–159, 2010.

Artur Back de Luca, Guojun Zhang, Xi Chen, and Yaoliang Yu. Mitigating data heterogeneity in federated learning with data augmentation. *CoRR*, abs/2206.09979, 2022. doi: 10.48550/ARXIV.2206.09979. URL `https://doi.org/10.48550/arXiv.2206.09979`.

Xiaofeng Fan, Yining Ma, Zhongxiang Dai, Wei Jing, Cheston Tan, and Bryan Kian Hsiang Low. Fault-tolerant federated reinforcement learning with theoretical guarantee. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang,

and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1007–1021. Curran Associates, Inc., 2021.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1510.00149`.

JiChu Jiang, Burak Kantarci, Sema Fatma Oktug, and Tolga Soyata. Federated learning in smart city sensing: Challenges and opportunities. *Sensors (Basel, Switzerland)*, 20, 2020. URL `https://api.semanticscholar.org/CorpusID:226251549`.

Peter Kairouz, Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, and et al. Advances and open problems in federated learning. arXiv:1912.04977v3, 2021.

Sai P. Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. arxiv:1910.06378v4, 2021.

Jakub Konečný, Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. arXiv: 1610.02527v1, 2016.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Transactions on knowledge and data engineering*, 35(4):3347–3366, 2021a.

Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. 05 2022. doi: 10.1109/ICDE53745.2022.00077.

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

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. arXiv: 1812.06127v5, 2020b.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6357–6368. PMLR, 18–24 Jul 2021b.

Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption. *International Journal of Intelligent Systems*, 37(9):5880–5901, 2022. doi: https://doi.org/10.1002/int.22818.

Priyanka Mary Mammen. Federated learning: Opportunities and challenges. arXiv: 2101.05428v1, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. arXiv:1602.05629v1, 2016.

Brendan McMahan, Jakub Konečný, Felix X. Yu, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492v2, 2017a.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1273–1282. PMLR, 2017b.

Aritra Mitra, Rayana Jaafar, George J. Pappas, and Hamed Hassani. Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14606–14619. Curran Associates, Inc., 2021.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Sashank Reddi, Zachary Charles, Zachary Garrett Manzil Zaheer, Keith Rush, Sanjiv Kumar, Jakub Konečný, and Brendan McMahan. Adaptive federated optimization. arXiv:2003.00295v5, 2021.

Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity. *IEEE Journal on Selected Areas in Information Theory*, 3(2):197–205, 2022.

Jonatan Reyes, Lisa Di Jorio, Cecile Low-Kam, and Marta Kersten-Oertel. Precision-weighted federated learning. arXiv:2107.09627v1, 2021.

Nuria Rodríguez-Barrosoa, Daniel Jiménez Lópeza, M. Victoria Luzónb, Francisco Herreraa, and Eugenio Martínez-Cámara. Survey on federated learning threats: concepts, taxonomy on attacks and defences, experimental study and challenges. arXiv:2201.08135v1, 2022.

Anit Sahu, Li Tian, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. 12 2018.

Mihye Seol and Taejoon Kim. Performance enhancement in federated learning by reducing class imbalance of non-iid data. *Sensors*, 23(3), 2023. URL `https://www.mdpi.com/1424-8220/23/3/1152`.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Sharan Vaswani, Francis Bach, and Mark Schmidt. Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, 16–18 Apr 2019.

Jianyu Wang, Qinghua Liu, Hao Liang, TGauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. 2020.

Jianyu Wang, Zachary Charles, and Al. A field guide to federated optimization. arXiv:2107.06917v1, 2021.

Aimin Yang, Zezhong Ma, Chunying Zhang, Yang Han, Zhibin Hu, Wei Zhang, and Xiangdong Huangand Yafeng Wu. Review on application progress of federated learning model and security hazard protection. *Digital Communications and Networks*, 9(1):146–158, 2023. ISSN 2352-8648. URL `https://www.sciencedirect.com/science/article/pii/S2352864822002474`.

Timothy Yang, Galen Andrew, Hubert Eichner, Wei Li Haicheng Sun, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning improving google keyboard query suggestions. arXiv:1812.02903v1, 2018.

Mang Ye, Xiuwen Fang, Bo Du, Pong C. Yuen, and Dacheng Tao. Heterogeneous federated learning: State-of-the-art and research challenges. arXiv:2307.10616v2, 2023a.

Mang Ye, Xiuwen Fang, Bo Du, Pong C. Yuen, and Dacheng Tao. Heterogeneous federated learning: State-of-the-art and research challenges. arXiv:2307.10616v2, 2023b.

Shui Yu and Lei Cui. *Security and Privacy in Federated Learning*. Springer Nature Singapore, 2023. URL `https://doi.org/10.1007/978-981-19-8692-5_5`.

Honglin Yuan, Warren Morningstar, Lin Ninga, and Karan Singhal. What do we mean by generalization in federated learning? arXiv:2110.14216v2, 2022.

Manzil Zaheer, Devendra Sachan, Sashank J. Reddi, Satyen Kale, and ASanjiv Kumar. Adaptive methods for nonconvex optimization. page 9815–9825, 2018.

Zhongchang Zhou, Fenggang Sun, Peng Lan, and Zhijun Wang. Improved federated average algorithm for non-independent and identically distributed data. In Jiande Sun, Yue Wang, Mengyao Huo, and Lexi Xu, editors, *Signal and Information Processing, Networking and Computers*, pages 1261–1267. Springer Nature Singapore, 2023.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Faculté des sciences
Place des Sciences, 2 bte L6.06.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/sc