MÁSTER EN TRATAMIENTO ESTADÍSTICO-COMPUTACIONAL DE LA INFORMACIÓN

TRABAJO FIN DE MÁSTER

Privacy Preserving Machine Learning

Javier Sempere Hernández

UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS
&
UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Tutor profesional: Juan Miguel Auñón García

Tutor académico: Carlos Gregorio Rodríguez

Madrid, Agosto 2024

Abstract:

This document introduces key topics related to Privacy Preserving Machine Learning (PPML) and Privacy Enhancing Technologies (PETs). The explained algorithms are applicable to specific use cases where data cannot or should not be shared freely. In particular, Federated Learning will be studied with a horizontal partition of data, examining the impact of statistical heterogeneity on model performance using a custom implementation of the algorithms. Additionally, it will be explored how to train models on vertically partitioned data and techniques to preserve data privacy through differential privacy and generative models. Each chapter will present the mathematical definition of the problem, analyze the results of the implementation, and discuss some limitations or considerations associated with each technique.

This work has been made possible thanks to the warm welcome I received at GMV, where I was allowed to study scientific articles from the beginning and was supported in my training. In particular, I want to thank my professional tutor Juan Miguel Auñón García and my colleague Daniel Hurtado Ramírez for the help they have provided me. I also want to thank my academic tutor Carlos Gregorio Rodríguez for his guidance during the course of this work. And especially, I want to thank my boyfriend Miguel, who has put up with me while balancing the master's studies, the work at GMV, and the writing of this document.

Contents

Chapter 1

Introduction

1.1 Privacy in the Big Data era

In today's interconnected world, data is often referred to as the new oil, a resource of immense value. The surge of big data has revolutionized industries, transforming the way businesses operate and decisions are made. From personalized marketing to predictive analytics, big data enables a level of insight previously unimaginable. However, alongside these benefits, the proliferation of data has brought significant concerns regarding privacy. As data collection and processing become more pervasive, the importance and necessity of privacy in the big data era cannot be overstated.
Big data refers to extremely large datasets that can be analyzed computationally to reveal patterns, trends, and associations, particularly relating to human behavior and interactions. The volume, velocity, and variety of data generated today are unprecedented, driven by the ubiquity of internet-connected devices, social media, and advanced sensors. Companies and governments harness this data to optimize operations, enhance customer experiences, and improve public services. While the benefits of big data are clear, they come at a cost. The more data is collected, the greater the risk of privacy breaches. Users often unknowingly trade their privacy for convenience, sometimes with far-reaching consequences. The loss of privacy due to big data can lead to several risks:

- **Identity theft and fraud:** Personal data, if accessed by malicious actors, can be used to steal identities and commit fraud. With detailed personal information, criminals can impersonate individuals to access financial accounts, apply for loans, or engage in other fraudulent activities.
- **Surveillance and Profiling:** Extensive data collection enables the creation of detailed profiles of individuals. Governments and corporations can use these profiles for surveillance, often infringing on civil liberties. The ability to monitor and predict individuals' actions poses a significant threat to personal freedom and autonomy.
- **Discrimination and bias:** Big data algorithms, often used in decision-making processes, can perpetuate and amplify biases present in the data. This can lead to discriminatory practices in areas like hiring, lending, and law enforcement. Ensuring data fairness and preventing algorithmic bias is crucial to maintain equity.
- **Loss of trust:** When organizations fail to protect personal data, they risk losing the trust of their customers. High-profile data breaches have shown that trust, once lost, is hard to regain. Consumers are increasingly aware of privacy issues, and companies must prioritize data protection to maintain their reputation.

1.1.1 What is Privacy-Preserving Machine Learning

Privacy Preserving Machine Learning (PPML) encompasses a range of techniques designed to train machine learning models while protecting the privacy of the data involved. PPML leverages cryptographic methods and algorithmic strategies to ensure that sensitive information is not exposed during the training or inference processes. In order to achieve this, several techniques are being developed:

- **Homomorphic encryption:** Allows computations to be performed on encrypted data without needing to decrypt it first. This ensures that data remains secure throughout the computation process.
- **Secure Multi-Party Computation:** Enables multiple parties to collaboratively compute a function (i.e. an algorithm) over their inputs while keeping those inputs private.
- **Differential privacy:** Adds noise to the data or the outputs of queries to prevent the identification of individuals within a dataset. It provides mathematical guarantees that individual data points cannot be distinguished.
- **Federated Learning:** A decentralized approach to machine learning where the model training occurs across multiple devices or servers (nodes) that hold local data samples. Instead of sharing raw data, only model updates (like gradients) are exchanged and aggregated to improve the global model.
- **Anonymization and Pseudonymization:** These techniques remove or alter personally identifiable information, reducing the risk of re-identification. Anonymization makes data non-attributable to an individual, while pseudonymization replaces private identifiers with fake ones.

1.1.2 EU and Spanish regulation

Both individuals and organizations have roles to play in protecting privacy. Individuals should be educated about privacy risks and empowered to make informed decisions about their data. This includes understanding the privacy policies of services they use and taking advantage of tools to manage their data preferences. Organizations, on the other hand, must adopt a proactive approach to privacy. This involves not only complying with legal requirements but also going beyond to implement best practices in data protection. Regular privacy impact assessments, robust data security measures, and transparent communication with users are essential.
In recent years, data governance has emerged as a critical area of focus both within the European Union (EU) and at the local level in Spain. These efforts are driven by regulatory initiatives and strategic frameworks aimed at maximizing the societal and economic benefits of data while ensuring transparency, accessibility, and a responsible management. The European Union has been proactive in shaping data governance policies to harness the potential of data-driven economies. A cornerstone of this effort is the EU Data Act, a legislative proposal introduced by the European Commission in 2023 [8].
The EU Data Act seeks to establish unified rules governing the access, sharing, and reuse of data generated across all economic sectors within the EU. Central to the objectives of the EU Data Act is the promotion of fairness in the digital environment. It aims to ensure equitable access to data. This access is intended to stimulate a competitive data market, foster innovation, and facilitate the development of new data-driven services and products. By clarifying data usage rights for consumers and businesses, the Act aims to enhance trust in digital services and support Europe's broader digital transformation goals. The Act addresses the underutilization of industrial data, which has been a significant challenge despite the exponential growth in data volume globally. By promoting data sharing and ensuring compliance with European data protection regulations such as the General Data Protection Regulation (GDPR), the EU seeks to unlock the economic potential of unused data. It is estimated that the EU Data Act could contribute substantially to GDP growth, positioning Europe as a leader in the global data economy by 2030. Complementing the EU Data Act is the EU Data Strategy, which provides a comprehensive framework for leveraging data to drive economic growth and societal progress. The strategy is built on Digital Principles that emphasize empowering

consumers and businesses with greater control over their data. By encouraging sustainable practices and supporting the development of a competitive data market, the EU Data Strategy aims to capitalize on the transformative potential of data across various sectors, including healthcare, agriculture, and transport.

Additionally, at the European level, there is a more general regulation on AI, the **EU AI Act** [35]. The EU AI Act is a proposed regulatory framework by the European Union aimed at ensuring the safe and ethical use of artificial intelligence across various sectors. The EU AI Act emphasizes that AI systems, especially high-risk ones, must comply with existing data protection laws, such as the General Data Protection Regulation (GDPR). This includes stringent data governance requirements to protect individuals' privacy rights. To prevent discrimination and ensure fairness, the Act requires that data used in AI systems be of high quality, relevant, and free from bias. Proper data handling is crucial for maintaining privacy and avoiding biased outcomes. The Act also encourages anonymization and data minimization practices, ensuring that AI systems only use the necessary amount of data without compromising individuals' privacy. Moreover, the Act mandates transparency in data usage, requiring AI providers to inform users about how their data is being use.

At the local level in Spain, the Spanish Federation of Municipalities and Provinces (FEMP) has implemented a Model Ordinance on Data Governance to standardize and enhance data management practices across municipal administrations [3]. This initiative is part of broader efforts to improve transparency, efficiency, and decision-making processes through effective data governance. The FEMP Model Ordinance on Data Governance highlights the strategic importance of data as a valuable asset for local administrations. It addresses various aspects of data governance, including data acquisition, management, exploitation, and openness. By promoting equal access to data and encouraging its reuse, the ordinance aims to maximize the benefits of data for local communities and support evidence-based policymaking. Aligned with the principles of the GDPR, the ordinance places a strong emphasis on data privacy and the anonymization of data to protect individual rights. It also recognizes the critical role of data in supporting emerging technologies such as artificial intelligence (AI). The ordinance advocates for the use of quality data to train AI models effectively, thereby enhancing the accuracy and reliability of AI-driven solutions implemented by local governments.

While the EU Data Act aims to harmonize data regulations across member states to facilitate cross-border data flows and innovation, local initiatives such as the FEMP Model Ordinance on Data Governance provide tailored approaches to address specific regional needs and challenges. The synergy between EU and local initiatives is crucial for promoting a cohesive approach to data governance that balances regulatory compliance with local autonomy and innovation. By aligning with EU standards and guidelines, local administrations can leverage best practices in data management and governance to enhance service delivery, improve citizen engagement, and optimize resource allocation.

1.1.3 Use cases

Although not all use cases of this technology can be listed, such as collaboration between state and international administrations, research laboratory collaboration, collaboration for detecting tax fraud, etc., we will mention some of the use cases where research has been most focused and practical implementations currently exist.

- **Healthcare sector:** One of the most promising applications of PPML is within the healthcare sector. Medical institutions collect vast amounts of sensitive patient data to improve diagnostics and treatment outcomes. However, the utilization of this data must be handled with extreme care to protect patient privacy. PPML techniques such as federated learning enable healthcare providers to collaborate on improving machine learning models without sharing raw patient data. This decentralized approach ensures that sensitive information remains secure within individual institutions while still benefiting from collective insights.

- **Financial services:** In the realm of financial services, institutions face rigorous regulatory requirements to safeguard customer information. At the same time, they seek to leverage customer data to personalize services and detect fraudulent activities. PPML techniques like differential privacy allow financial institutions to aggregate customer data while obscuring individual identities. This ensures compliance with privacy regulations without compromising the utility of data-driven insights.
- **Smart cities and IoT:** As cities become more interconnected through the Internet of Things (IoT), vast amounts of data are generated from sensors and devices. Smart city initiatives rely on this data to optimize urban planning, transportation systems, and public services. PPML plays a critical role in preserving citizen privacy by anonymizing data streams. By implementing techniques such as homomorphic encryption, cities can analyze encrypted data without decrypting it, thereby protecting the privacy of individuals while extracting valuable insights for urban development.
- **E-commerce:** E-commerce platforms thrive on understanding consumer behavior to offer personalized recommendations and targeted advertisements. However, this reliance on user data raises concerns about privacy infringement. PPML solutions like multi-party computation enable e-commerce companies to collaborate with data providers and advertisers without directly exposing individual user preferences. By jointly analyzing encrypted data, these platforms can tailor offerings to user interests while preserving user anonymity.

1.2 How this document is structured

In the remainder of Chapter 1, the basic elements for neural network models that will be used later will be briefly introduced.In particular, the Multilayer Perceptron and Convolutional Neural Networks will be briefly introduced, as they will be used throughout the rest of the work. The decision to study neural networks rather than other ML models is due to the fact that most current research in Privacy Enhancing Technologies focuses on these models.

Chapter 2 will provide a detailed study of federated learning, particularly on horizontally partitioned data and under a cross-silo topology. The objective of this chapter is twofold: to introduce federated learning and to examine its performance under different Non-IID data sampling schemes, which is one of the main challenges of this technology. For this purpose, the necessary code has been developed to freely experiment with algorithms and data sampling. This code can be found at: `www.github.com/JSempereH/TFM`.

Chapter 3 will discuss how models can be trained on vertically partitioned data, aligning datasets with cryptographic protocols such as Private Set Intersection and training neural networks using the SplitNN algorithm. Again, an implementation of this can be seen in the project repository.

Chapter 4 covers two technologies that can be used to protect data privacy in specific cases. The topics covered require a very extensive formal introduction, so only the basic concepts needed to understand the use cases and challenges will be explained. First, differential privacy is introduced along with some algorithms that utilize it. Next, it examines how generative models can be used to protect data privacy, specifically for tabular data. To this end, one of the most commonly used techniques, CTGANs, is studied and implemented on sample data to assess its performance.

The topics covered in these chapters are closely related to several subjects of the master's program, such as Neural Networks, Optimization Techniques, Pattern Recognition, Data Mining, or Computer Tools for Big Data. Finally, this work concludes with a summary of the topics covered, the challenges encountered, and a personal reflection on the technologies studied and their current relevance.

1.3 Multilayer Perceptrons

Since most current research and practical implementations are being conducted in the field of deep learning, a large part of the algorithms explained in this work will focus on this area. Therefore, some basic pattern classification algorithms that will be used in this work will be introduced.
A Multilayer Perceptron (MLP), also known as feedforward neural network, is a mathematical model mainly used in supervised learning. Essentially, an MLP is a directed acyclic graph which represents the composition of functions. Each function or **layer** is a collection of neurons. A neuron is defined as:

$$y = f(\mathbf{x}; \theta) = \omega^T \mathbf{x} + b, \qquad \theta = (\omega, b) \tag{1.1}$$

The term *perceptron* refers to a linear classifier consisting of one layer [36], $f(\mathbf{x}; \theta) = \mathbb{I}(\omega^T \mathbf{x} + b > 0)$. Here, $\mathbb{I}(a > 0)$ is the Heaviside function, which is non-differentiable. In MLP, the activation function from the original perceptron $\mathbb{I}$ is usually replaced by another differentiable function $\psi \colon \mathbb{R} \to \mathbb{R}$. The internal layers of MLP are usually named **hidden layers**, the last one is called **output layer**. The dimensionality of the hidden layers determines the **width** of the neural network [17]. Each layer l consists of many units $\mathbf{z}_l$ which are computed as a linear transformation of the units from the previous layer $l-1$ passed element-wise through the activation function [29]:

$$\mathbf{z}_l = \psi_l(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l)$$

We can choose different activation functions that will define our model and impact in the performance of the training. If we use a linear activation function $\psi_l(x) = K_l x$ then our neural network becomes just a linear model [29], that's why usually non-linear activation functions are used, since we would like to capture non-linear relationships between the input data and the output. Also, the Universal Approximation Theorem states that a neural network with a single hidden layer and non-linear activation functions can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to any desired degree of accuracy. Since the goal of an MLP is to approximate some function f^* (for example, for a classifier $y = f^*(\mathbf{x})$), non-linear activation functions are necessary to achieve this level of approximation. However, they can be used when we know the problem is linearly separable (think of two separate clusters of points in $\mathbb{R}^2$ in a classifier model) or in the output layer, for example when the MLP acts as a regression model.
If we change the Heaviside function by the sigmoid (logistic) function $\sigma(x) = \frac{1}{1+e^{-x}}$ we get a smooth approximation of $\mathbb{I}$. Another choice of activation function could be $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ which has a similar shape but its image is $(-1, 1)$. Both are valid activation function and have been used. However, since we need to compute gradients in the optimization process in order to update the weights, a close to $\mathbf{0}$ gradient would be a problem, since it will make hard to train a model using gradient descent (vanishing gradient problem). Both functions have an almost horizontal slope (gradient near 0) for large positive and negative inputs. Also, $\|\frac{d}{dx}\sigma(x)\| = \|\sigma(x)(1-\sigma(x))\| \leq \frac{1}{4} < 1$, therefore several multiplications will quickly approximate to zero. Although they are used in practice (for example, $\sigma(\cdot)$ is used in the output layer for binary regression problems), in order to train deep models we need non-saturating activation functions for the hidden layers. One of the most used is *rectified linear unit*: $ReLU(x) = \max(a, 0) = a\mathbb{I}(a > 0)$. The gradient of $ReLU'(z) \neq 0$ as long as z is positive: this function don't require input normalization to prevent them from saturating. As stated in [16], the rectifier activation function allows a network to easily obtain sparse representations, the *hard* saturation (gradients being exactly 0) help supervised training: experimental results suggest that networks with ReLU activation functions in the hidden layers have better convergence performance than using sigmoid [23].

The neural network tries to approximate the target function $\mathbf{y} = F(\mathbf{x})$, the *true* relation between the variables. The network, $f(x; \theta)$, *learns* by searching the parameters $\boldsymbol{\theta}$ that minimizes a loss function $J(\boldsymbol{\theta})$, which measures the distance between the output and the target or the proximity between probability densities of random variables [6]. If we consider a k-class classification problem with $\mathcal{X} \subset \mathbb{R}^d$ the feature space and $\mathcal{Y} = \{1, \ldots, k\}$ the label space, given the dataset $D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$ and a MLP $f \colon \mathcal{X} \to \mathbb{R}^k$ with a softmax as the output layer. For any loss function $J(\boldsymbol{\theta}) = J(f(\mathbf{x}, \boldsymbol{\theta}), y)$, assuming there is a joint distribution $P(\mathbf{x}, y)$ over $\mathcal{X}$ and $\mathcal{Y}$, the **risk** of f is defined as $R_J(f) = \mathbb{E}\Big[J(f(\mathbf{x}; \boldsymbol{\theta}), y)\Big] = \int J(f(\mathbf{x}; \theta), y) dP(\mathbf{x}, y)$ and the

empirical risk is defined as $\hat{R}_J(f) = \mathbb{E}_D\Big[J(f(\mathbf{x};\boldsymbol{\theta}), y)\Big]$ assuming that $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are IID samples from $P(\mathbf{x}, y)$. Since the nonlinearity of an MLP makes the loss function to become non-convex, it's trained using iterative, gradient-based optimizers. Most neural networks, and in particular the models that will be used in this work, are trained using maximum likelihood: the loss function is the negative log-likelihood [1] [17]. That is, we will compute the *cross-entropy*[2] between the training data and the model distribution:

$$J(\theta) = -\mathbb{E}_{\mathbf{x},y\sim D} \log_{P(\mathbf{x},y)}(y|\mathbf{x}) \tag{1.2}$$

In a k-class classification model with a cross-entropy loss function, the empirical risk to minimize is $\hat{R}_J(f) = -\frac{1}{n}\sum_{i=1}^n \sum_{j=1}^k \mathbf{y}_{ij} \log f_j(\mathbf{x}_i;\boldsymbol{\theta})$, where $\mathbf{y}_{ij}$ is the j-th element of the one-hot encoded label of $\mathbf{x}_i$ such that $\mathbf{1}^T\mathbf{y}_i = 1$, $\forall i$, and f_j is the j-th element of the softmax output layer of f [55].

1.4 Optimizer

Typically, the minimization of $J(\boldsymbol{\theta})$ is achieved using a gradient descent algorithm. Large training datasets are good for generalization, but since $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \frac{1}{n}\sum_{i=1}^n \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x};\boldsymbol{\theta}), y)$ has a computational complexity of $\mathcal{O}(n)$ for each step [17], this approach is prohibitive for n in a scale of millions. Optimization algorithms that use the whole dataset are called **batch**[3] gradient methods. When only a single sample is used each step, the method is called **stochastic**. The **Minibatch Stochastic Gradient Descent** (MSGD) algorithm (and its variants) is preferred, since the gradient direction in SGD oscillates because of the additional noise added by random sampling [44]. The MSGD algorithm makes and estimation of the gradient selecting randomly a *minibatch* $b = \{(\mathbf{x}, y)^{(1)}, \cdots, (\mathbf{x}, y)^{(m)}\}$, $m < n$ such as:

$$\mathbf{g} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^m l(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)}) \tag{1.3}$$

Each step of the MSGD algorithm, the parameters $\boldsymbol{\theta}$ are updated given a learning rate $\eta > 0$: $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\mathbf{g}$. A constant learning rate during training may lead to a zone where the gradient estimate "jumps around" a local minimum, although if the network is complex enough to represent the underlying function, a constant learning rate usually works well in practice [43]. In some implementations, the learning rate is updated during training, an example could be multiplying by a factor $0 < \gamma < 1$ each T number of steps[4]. Here, we encounter some challenges [37]:

- A small η leads to slow convergence of vanilla MSGD, while if it's too large the loss function will fluctuate around the minimum or diverge.

- A learning rate scheduler needs to be defined at the beginning and it's not adapted to dataset's characteristics.

- Since $J(\boldsymbol{\theta})$ is non-convex (recall that affine transformations followed by an activation function such as ReLU is non-covex). A key challenge is avoiding getting trapped in suboptimal local minima. Also, according to [9], the difficulty comes from saddle points present in high-dimensional non-convex optimization that are hard for SGD to scape, since the gradient is close to zero in all dimensions.

[1] Empirical risk minimization is equivalent to Maximum Likelihood Estimation when the risk is defined as the negative of the likelihood. Note that ERM does not limit itself to that particular class of risk functions.

[2] This is valid for a loss consisting of a negative log-likelihood, not only for the negative log-likelihodd of a softmax distribution.

[3] The term *batch* may also refer to a subset of the training set. The size of a minibatch is usually called *batch size*.

[4] This is actually what the *StepLR* class does in the popular Python package torch.optim.

There are several gradient-based variants that try to tackle these challenges. For example, a momentum term can be used to help accelerate MSGD and dampen oscillations [33]. Momentum stores the gradient of the past step to adjust the new direction:

$$\begin{aligned} \mathbf{v}_t &= \alpha \mathbf{v}_{t-1} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \eta > 0,\ \alpha \in [0,1] \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \mathbf{v}_t \end{aligned} \tag{1.4}$$

The momentum term increases the updates for dimensions whose gradients point in the same direction, allowing the algorithm to build momentum and move more efficiently towards the minimum. Conversely, it reduces updates for dimensions whose gradients change direction, dampening out oscillations and preventing the algorithm from getting stuck in local minima [37].

Another variant is **Nesterov accelerated gradient** [49], which is an extension of Gradient Descent with momentum where the computation of the gradient is performed using projected parameters and not the actual values:

$$\begin{aligned} \mathbf{v}_t &= \alpha \mathbf{v}_{t-1} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} - \alpha \mathbf{v}_{t-1}) \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \mathbf{v}_t \end{aligned} \tag{1.5}$$

Here, similar to GD with momentum, $\mathbf{v}_t$ is the direction and speed at which the parameters should be tweaked and α determines how quickly the previous gradients will decay. Nesterov Accelerated Gradient tries to tackle the exploding gradient case: where the velocity added to the parameters gives an unwanted high loss. In this case, if the velocity update lead to a bad loss, the gradients will redirect the update back to $\boldsymbol{\theta}$.

Still, with the previous algorithms the hyperparameters η and α need to be fixed or changed during training a heuristic way. With the **Adaptative Gradient Algorithm** (**Adagrad**), the learning rate change depending on the parameters: it performs larger updates for infrequent parameters and smaller updates for frequent parameters. That's it, each parameter is updated with a different learning rate each step. We set $g_{t,i}$ the gradient estimate w.r.t the parameter θ_i at step t. The Adagrad update rule is:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} g_{t,i} \tag{1.6}$$

Where $G_{t,i}$ is the sum of squares of the gradients w.r.t θ_i up to step t, and $\epsilon > 0$ is the decimal smallest number in the machine, in order to avoid division by zero. With Adagrad, the selection of η is irrelevant since the term will be scaled by the root. Most implementations just fix it at 0.01. The problem with this algorithm is that the accumulated sums will increase during training and eventually the learning rate will be close to zero, therefore the parameters are not updated at a certain point. In order to fix this, **Adadelta** restricts the number of accumulated gradients to some fixed size w [53]. Since storing w previous squared gradients is inefficient, it was proposed to compute an exponentially decaying average of the squared gradients, given a decay constant $\rho \in (0,1)$, similar to that used in Momentum. So, let $\mathbf{g}_t$ be the estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$, $\mathbf{v}$ and $\mathbf{u}$ the square average and the accumulated variable resp., both initialized at $\mathbf{0}$ and $\lambda > 0$ the weight decay, the Adadelta algorithm is:

$$\begin{aligned} \mathbf{g}_t &\leftarrow \mathbf{g_t} + \lambda \boldsymbol{\theta}_{t-1} \\ \mathbf{v}_t &\leftarrow \mathbf{v}_{t-1}\rho + \mathbf{g}_t^2(1-\rho) \\ \Delta \mathbf{x}_t &\leftarrow \frac{\sqrt{\mathbf{u}_{t-1} + \epsilon}}{\sqrt{\mathbf{v}_t + \epsilon}} \mathbf{g}_t \\ \mathbf{u}_t &\leftarrow \mathbf{u}_{t-1}\rho + \Delta \mathbf{x}_t^2(1-\rho) \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \gamma \Delta \mathbf{x}_t \end{aligned} \tag{1.7}$$

Where the operations between vectors like the square is elementwise, i.e. $\mathbf{g}^2 = \mathbf{g} \odot \mathbf{g}$.
Another first-order gradient-base optimization algorithm for stochastic objective functions that is computationally efficient and compares favorably to other adaptive learning-method algorithms is **Adaptive Moment Estimation** (**Adam**) [22]. In addition to storing an exponentially decaying average of squared gradients $\mathbf{v}_t$ like Adadelta, this algorithm also keeps an exponentially decaying average of past gradients $\mathbf{m}_t$, similar to momentum. Let $\beta_1, \beta_2 \in [0, 1)$ be the exponential decay rates for the moment estimates:

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}) \\
\mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \quad \text{(Biased first moment estimate)} \\
\mathbf{v}_t &\leftarrow \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \quad \text{(Biased second raw moment estimate)} \\
\hat{\mathbf{m}}_t &\leftarrow \mathbf{m}_t / (1-\beta_1^t) \quad \text{(Bias-corrected first moment estimate)} \\
\hat{\mathbf{v}}_t &\leftarrow \mathbf{v}_t / (1-\beta_2^t) \quad \text{(Bias-corrected second raw moment estimate)} \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \hat{\mathbf{m}}_t / \left(\sqrt{\hat{\mathbf{v}}_t} + \epsilon\right)
\end{aligned} \tag{1.8}$$

According to [22], good default settings (in their machine learning experiments) are $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

In order to use this gradient-based algorithms we need to compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, which is done using the **Backpropagation** algorithm [38], which leverages the composite structure of the neural network to efficiently compute the gradient [7].

1.5 Convolutional Neural Networks

When the input data is an image, a better approach for learning about $p(y|\mathbf{x})$ is to use Convolutional Neural Networks (CNN), since MLP tend to struggle with the computational complexity required to compute image data and the width required to train this kind of data may lead to overfitting [30]. CNNs are comprised of three types of layers: convolutional layers, pooling layers and fully-connected layers (which are essentially a MLP).

The convolution operation is the fundamental building block of a CNN. It involves a filter (or kernel) that is applied to the input image to produce a feature map. Mathematically, the convolution operation for a 2D input image I and a filter K can be expressed as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

Here, $I(i-m, j-n)$ represents the pixel value of the image at position $(i-m, j-n)$, and $K(m,n)$ is the weight of the filter at position (m,n). The resulting feature map $S(i,j)$ highlights the presence of specific features in the input image.

After the convolution operation, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU), defined in the previous section. This will be the activation function used in the models implemented in Section 2 and 3.
Pooling layers are used to reduce the spatial dimensions of the feature maps, thereby reducing the number of parameters and computation in the network. The most common type of pooling is max pooling, which selects the maximum value from a patch of the feature map. For a 2D feature map F, max pooling can be defined as:

$$P(i,j) = \max_{0 \le m < p, 0 \le n < p} F(i+m, j+n) \tag{1.9}$$

where p is the size of the pooling window. After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers (see Section 1.3). For more information on convolutional network architectures, see [30].

Chapter 2

Federated Learning

2.1 Introduction

Let's assume we are a hospital that wants to train a predictive model for skin diseases. In the patient database, there are records of some diseases, but perhaps not all the diseases we would like to study. It is possible that other research centers or hospitals have more records of diseases, which may be: common to the diseases present in our database or new diseases that we do not have records for. If it were possible to train a model with both, we would gain two potential advantages: we would have a larger number of cases for each disease (something necessary in most deep learning models) and the model could classify a greater number of diseases.
Of course, this is not so simple. Medical records are very sensitive and cannot be used (or shared) freely. For such cases, it might make sense to use Federated Learning, a novel approach to model training that allows machine learning models to be trained collaboratively across multiple decentralized devices or servers without exchanging raw data [28].

Unlike traditional centralized approaches, where data is collected and processed in a central server, FL enables training on local devices following a scheme of decentralized model training. This decentralization ensures that data remains on the device, granting a certain degree of privacy. In a centralized setting, the trained model is updated based on the complete dataset, which is stored in a unique server. In the federated setting, data is distributed across local devices (parties or clients) and training happens locally. This can be problematic, since the source of the data differ, the data can differ in various ways: unbalanced datasets, different distributions, etc. This is one of the main challenges of FL: **non-IID data**, since this will negatively affect the performance of the model [25], [56], [24].
Horizontal Federated Learning (HFL) and Vertical Federated learning (VFL) are two variations of the federated learning paradigm that differ in how they distribute and collaborate on data.

- **HFL:** Each party has a portion of the overall dataset, each party holds a different subset of samples but for the same features.
- **VFL:** The data is vertically partitioned, each party has different features for the same set of samples.

ID	Name	Occupation	Salary	Age	Genre
1111	Juan	Engineer	37000	45	Male
2222	Ana	Scientist	34500	28	Female
3333	Daniel	Plumber	28000	33	Male

Dataset from Node 1

ID	Name	Occupation	Salary	Age	Genre
4444	María	Mathematician	32000	26	Female
5555	Carla	Engineer	40000	50	Female

Dataset from Node 2

Figure 2.1: Example of HFL. Each node has different samples but with the same attributes.

ID	Name	Occupation	Salary
1111	Juan	Engineer	37000
2222	Ana	Scientist	34500
3333	Daniel	Plumber	28000
4444	María	Mathematician	32000
5555	Carla	Engineer	40000

ID	Age	Genre
1111	45	Male
2222	28	Female
3333	33	Male
4444	26	Female
5555	50	Female

Figure 2.2: Example of VFL. Each node has common samples but with different attributes. It may happen that not all elements are common or aligned.

HFL and VFL are not mutually exclusive, in some cases a combination of both schemes may be applied. In this chapter we will focus on HFL. Also, we will only study *Cross-Silo Federated Learning (Cross-Silo FL)*, which is a variation of FL that addresses the scenario where data is distributed across different organizations, usually few parties, each maintaining control over its own data. This setting is particularly relevant in industries where different organizations need to collaborate on machine learning task, such as healthcare (hospitals collaborating on medical research), finance (banks collaborating on fraud detection), epidemiological studies (international public health agencies studying disease spread), smart cities (urban planning authorities collaborating on public services optimization), etc. Ensuring interoperability between different silos is a huge challenge, since there needs to be a fixed standard in data format, structures and processing capabilities accross different organizations.
Another flavour of FL is *Cross device FL*, which was the original topology proposed [28]. In this type of setting, the scale of the parties is potentially much bigger (order of millions) since it's aimed to mobile devices, IoT, apps... where the data format and architecture is usually already defined by an organization. For example, this is the kind of FL that Google uses for training Gboard's models [54]. With *Cross device FL*, it's needed to take into account some technical difficulties like client selection strategy [41], connection overheard, connection dropouts, device heterogeneity, etc. Some of these challenges are shared with *Cross-Silo FL*. The main differences are the scale of parties, the computing resources (mobile devices vs data centers or computers) and the partition of data: *cross device* tends to be HFL while *cross-silo* could be HFL or VFL. Since we are focusing in *cross-silo FL*, it won't be studied the implications of client selection (since we are assuming the parties are few and well-defined), the connection overheard or dropouts. The focus of this work will be mainly the statistical heterogeneity challenge.
We will begin by studying the FedAvg algorithm [28], which is de facto approach for Federated Learning (FL). We will establish notation, examine some of its properties, and explore issues that arise when data is not independently identically distributed (statistical heterogeneity). Following that, various approaches that

have been proposed to address this problem will be developed, and the performance of each will be analyzed across different training architectures.

2.2 Objective function

Let $D = \{(\mathbf{x}, y)\}$ be the global dataset[1] and $D^i \subset D$ the i-th party's local dataset, $i = 1, ..., N$. Let ω_g^t and ω_i^t be the global model and the local model of the i-th party in round $t \in \{1, ..., T\}$, respectively. Since we are working in a *Cross-silo FL* setting, the main differences between a federated optimization scheme and a distributed one would be the unbalanced data and non-IID data.
In a general FL optimization framework, we want to minimize the objective function:

$$F(\omega) = \mathbb{E}_{i\sim D}\Big[F_i(\omega_g)\Big], \qquad F_i(\omega_g) = \mathbb{E}_{z\sim D^i}\Big[l_i(\omega_g, z)\Big] \tag{2.1}$$

Here, $l_i(\omega_g, z)$ represents the local loss function of client i (with local dataset D^i). As stated in [45], the objective function in 2.1 can take the form of an empirical risk minimization objective function:

$$F(\omega_g) = \sum_{i=1}^{N} \alpha_i F_i(\omega_g) \text{ where } F_i(\omega_g) = \frac{1}{|D^i|} \sum_{z\in D^i} l_i(\omega_g, z) \text{ and } \sum_{i=1}^{N} \alpha_i = 1 \tag{2.2}$$

If $\alpha_i = \frac{|D^i|}{\sum_{i=1}^{N} |D^i|}$, the objective function in 2.2 would be the empirical risk minimization objective function of $D = \cup_i D^i$. We recall that 2.1 is an optimization problem that could be solved using Stochastic Descent: $\omega_g^{t+1} = \omega_g^t - \eta_t \nabla F(\omega_g^t)$ where η_t is the learning rate at time t. In practice, we will use an unbiased estimator of the gradient of the local loss $g_i(\omega_g^t)$ such that $\mathbb{E}_{z\sim D^i}\Big[g_i(\omega_g^t)\Big] = \nabla F_i(\omega_g^t)$ using Stochastic Gradient Descent (SGD).

2.3 Statistical heterogeneity

In a real-world scenario, the data distributed among parties is Non-IID, the distribution from each party differs from the global distribution. Each client has its own local optima, which may be very different from the actual global optima. Under a IID setting, the global optima is *close* to the local optima, but under a Non-IID settig, there is a *drift* in the local updates, especially if the number of local epochs changes between parties.

In order to simulate a Non-IID setting, it has been chosen the same strategy as [24], the datasets will be partitioned into multiple subsets. Let the local data distribution be $P(x_i, y_i) = P(x_i|y_i)P(y_i) = P(y_i|x_i)P(x_i)$. From a distribution perspective, the following Non-IID cases summarized in [21] are considered:

- **Label distribution skew:** Different $P(y_i)$ across parties.
- **Feature distribution skew:** Different $P(x_i)$ across parties.
- **Same label but different features:** Different $P(x_i|y_i)$ across parties.
- **Same features but different labels:** Different $P(y_i|x_i)$ across parties.
- **Quantity skew:** Although the parties have the same $P(x_i, y_i)$, $|D^i|$ differs.

[1]Here, the global dataset is the union of the different local datasets, $D = \cup_{i=1}^{N} D^i$. In practical cases, there is no such dataset in order to ensure data privacy. However, we will consider it to conduct a performance study of the various algorithms.

For the **label distribution skew**, it will be used two settings. The first one, is a quantity-based label imbalance, where each party possesses data samples corresponding to a fixed number of labels. An extreme case is where each party only has data from a unique label.

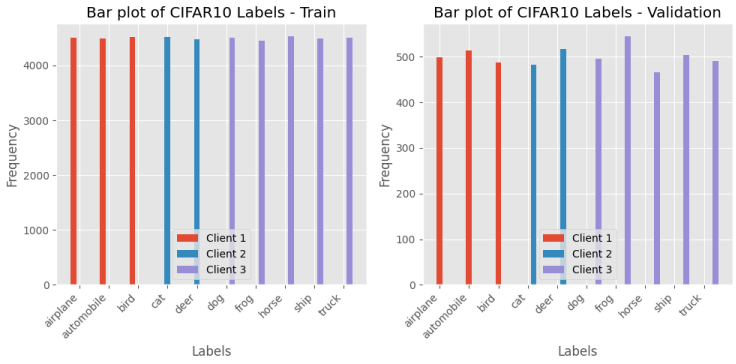

Figure 2.3: Example of label distribution skew with 3 clients, each one holding data from different labels.

The second setting is a distribution-based label imbalance, where the Dirichlet distribution $Dir(\boldsymbol{\beta})$ is used to assign different proportion of each label across parties. Given N parties, the probability density function from the Dirichlet distribution is defined as:

$$f(\mathbf{x}, \boldsymbol{\beta}) = \frac{1}{\boldsymbol{\beta}} \prod_{i=1}^{N} x_i^{\beta_i - 1}, \quad \sum_{i=1}^{N} x_i = 1, \; x_i \in [0, 1], \; \beta_i > 0$$

where the normalizing constant $B(\boldsymbol{\beta})$ is the multivariate beta function.

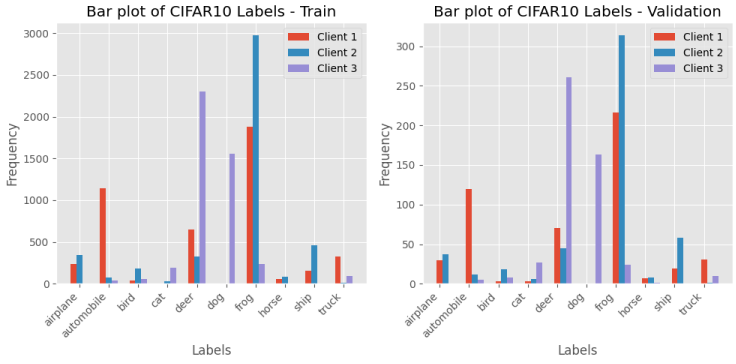

Figure 2.4: Example of label distribution skew with 3 clients using the Dirichlet distribution with $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$.

For the **feature distribution skew**, it will be used a noise-based feature imabalance, where different levels of Gaussian noise will be added for each party. Given $\sigma > 0$, for party i, a noise $N(0, \sigma \frac{i}{N})$ will be added to its local features. Increasing σ will increase the dissimilarity among parties. Note that although $P(x_i)$ differs, $P(y_i|x_i)$ is the same across parties.

It will be not considered the case where $P(y_i|x_i)$ differs, because then a distributed learning setting would make no sense (same feature should be classified as a different label across parties). Different $P(x_i|y_i)$ will neither be considered since we are focusing in a Horizontal FL setting. For the quantity skew, it will be used $Dir(\boldsymbol{\beta})$ to assign different proportion of the whole dataset to each party.

Figure 2.5: Example of feature distribution skew with $\sigma = 0.3$

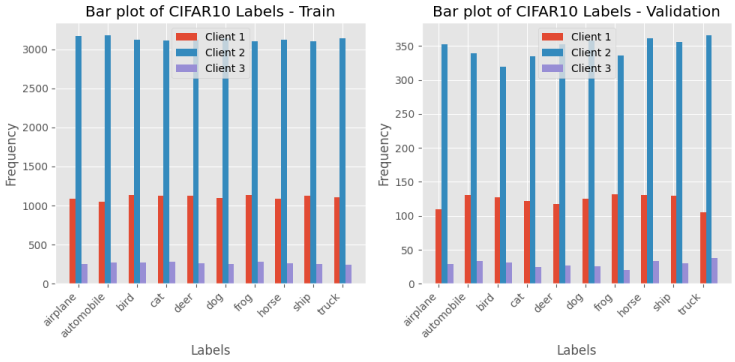

Figure 2.6: Example of quantity skew with 3 clients using the Dirichlet distribution with $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$.

For the rest of the chapter, these Non-IID Data will be applied in order to see the effect in the model's accuracy. The model used is a very simple[2] CNN described in Figure 2.7

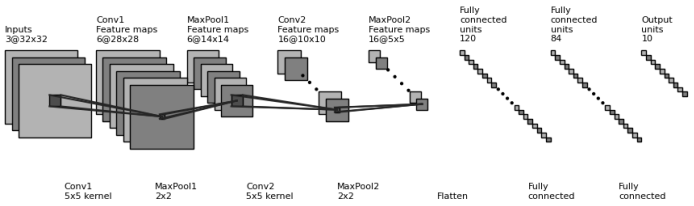

Figure 2.7: CNN used for all the following experiments.

2.4 FedAvg

In [28], it was introduced the **FederatedAveraging** algorithm (FedAvg).

[2]The aim of these experiments is to check the accuracy degradation between different Non-IID settings. It is not necessary to train the best model in order to achieve a high test accuracy, since the IID settings will serve as a baseline to compare between Non-IID cases. Note that the number of possible experiments combining all these settings is huge, and the whole project has been made with a low-spec laptop without a GPU.

Algorithm 1 FedAvg

Input: local datasets $D^i \; \forall i \in \{1, \dots, N\}$, number of parties N, number of communication rounds T, number of local epochs E, learning rate η for SGD, local mini-batch size B.
Output: global model ω_g^T.

procedure SERVER EXECUTION
  Initialize ω_g^0
  for round $t = 1, \dots, T$ **do**
    S_t (Selection of clients)
    for client $k \in S_t$ **in parallel do**
      $\omega_k^{t+1} \leftarrow ClientUpdate(k, \omega_g^t)$
    $\omega_g^{t+1} \leftarrow \sum_{k \in S_t} \frac{|D^k|}{\sum_{k \in S_t} |D^k|} \omega_k^{t+1}$
procedure $ClientUpdate(k, \omega_g^t)$
  $\omega_k^t \leftarrow \omega_g^t$
  $\mathcal{B} \leftarrow$ Batches of D^k of size B
  for local epoch $i = 1, \dots, E$ **do**
    for batch $\mathbf{b} \in \mathcal{B}$ **do**
      $\omega_k^t \leftarrow \omega_k^t - \eta \nabla l(\omega_k^t; \mathbf{b})$
  return ω_k^t to the server.

As we see from Algorithm 1, *FedAvg* took into account a client selection strategy. Since we are focusing in a framework where the clients are limited, we will not consider any subset of clients each round: all the clients will participate.
The algorithm is quite simple, the server is averaging the local models' weights, with a constant parameter determining the contribution of each client given the size of the local dataset.
In order to test the performance of FedAvg in different Non-IID settings, it has been used the CNN architecture described in Figure 2.7, using CIFAR10 dataset and with a hyperparameter selection based on [24]. First, the local metrics for each client is shown: these metrics are computed on D^i, for $i = 1, 2, 3$. With each setting, D^i changes and also its cardinal. How the data is partitioned is explained in section 2.3. We begin with a IID setting in a 50 communication rounds FL training, each client performs only 2 local epochs and after the aggregation step, each client evaluates its model on the local test dataset, which has a similar distribution as the local train dataset (local test dataset represents the 10% of D^i).

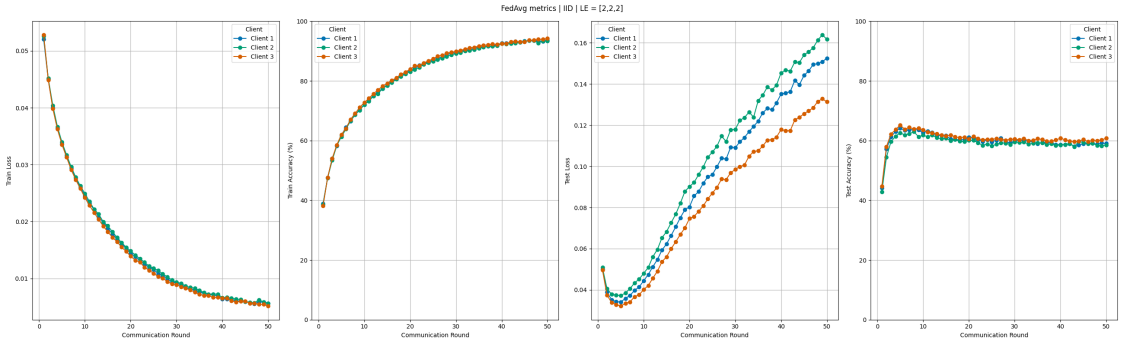

Figure 2.8: Local metrics for 3 clients in 50 communication rounds using FedAvg with a IID setting over the CIFAR10 dataset.

From Figure 2.8 we can see that the model converges with a local accuracy 60% for all clients and the model's behavior is similar between clients. Next experiment will simulate a Non-IID setting with label

distribution skew (see Figure 2.4) with $\boldsymbol{\beta} = (1, 1, 1)$ and 2 local epochs for each client. Given the randomness from sampling, the label distribution can change significantly between clients:

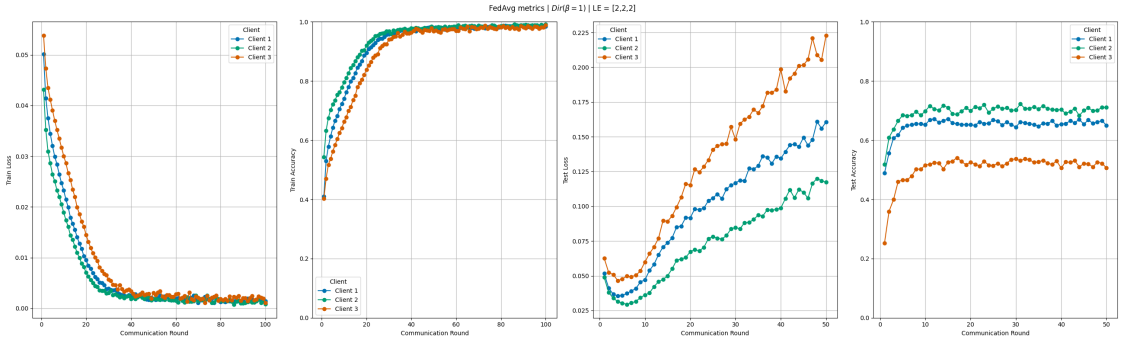

Figure 2.9: Local metrics for 3 clients in 50 communication rounds using FedAvg with a Non-IID setting over the CIFAR10 dataset. Label distribution skew using the Dirichlet distribution with $\boldsymbol{\beta} = (1, 1, 1)$

From Figure 2.9, we can see that different $P(y_i)$ between clients could deteriorate the model performance, in this example, the third client had a very different distribution from the first and second client. In the aggregation step, the model deteriorates since it has been trained with a very different loss landscape. In the next experiment, the Non-IID setting will be simulated with a Quantity skew using the Dirichlet distribution $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$, see Figure 2.6 for an example of this data partition.

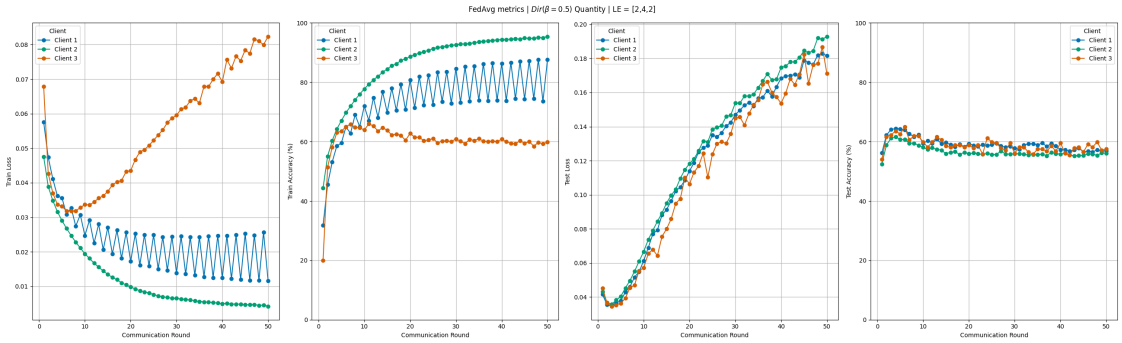

Figure 2.10: Local metrics for 3 clients in 50 communication rounds using FedAvg with a Non-IID setting over the CIFAR10 dataset. Quantity skew using the Dirichlet distribution with $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$

In Figure 2.10, the client with more samples (client 2) has a steady training progress, while the clients with less samples (client 1 and 3) have more errors in training step, although all clients have similar performance while testing its model, therefore, the clients with less data may see an improvement participating in a FL framework when the distribution $P(x_i, y_i)$ remains the same across clients but $|D^i|$ differs.
Now, we'll how FedAvg handles a situation where $P(y_i)$ is different across clients (label distribution skew), see Figure 2.3 for an example of this data partition. In the experiment, the first client has the first 2 labels, the second client the next 3 labels and the third client the remaining 5 labels.

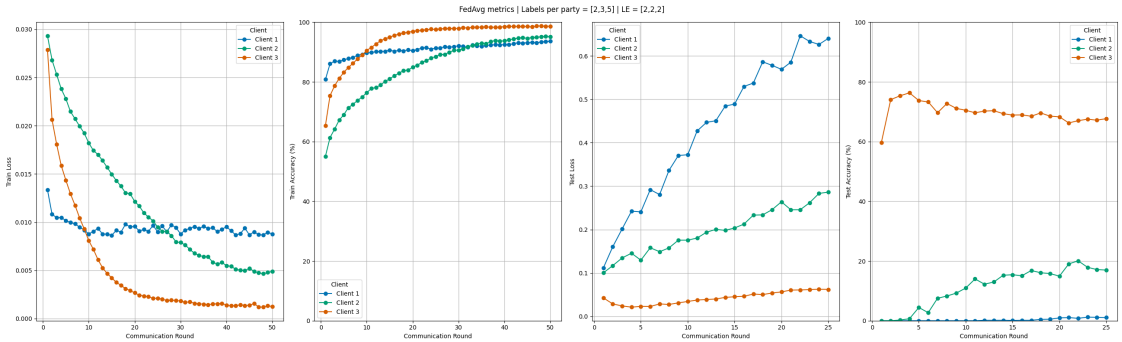

Figure 2.11: Local metrics for 3 clients in 50 communication rounds using FedAvg with a Non-IID setting over the CIFAR10 dataset. Label distribution skew, the first client data from 2 classes, the second client from 3 classes and the third client from the 5 remaining classes.

As we can see from Figure 2.11, the more labels a client has the better the performance. Since client 1 only has 2 labels, the aggregated global model doesn't classify correctly in its target domain. Recall from Algorithm 1 that each client's weight is proportional to its dataset's cardinal. Therefore, since client 1 has fewer data samples, this client's weights from the locally trained model looses importance against the others models.
For the next example, we will use the feature distribution skew, see Figure 2.5 for an example of this kind of assymetry using a Normal distribution $N(0, \sigma = 0.3)$. For the experiment, it will be used a $\sigma = 0.5$.

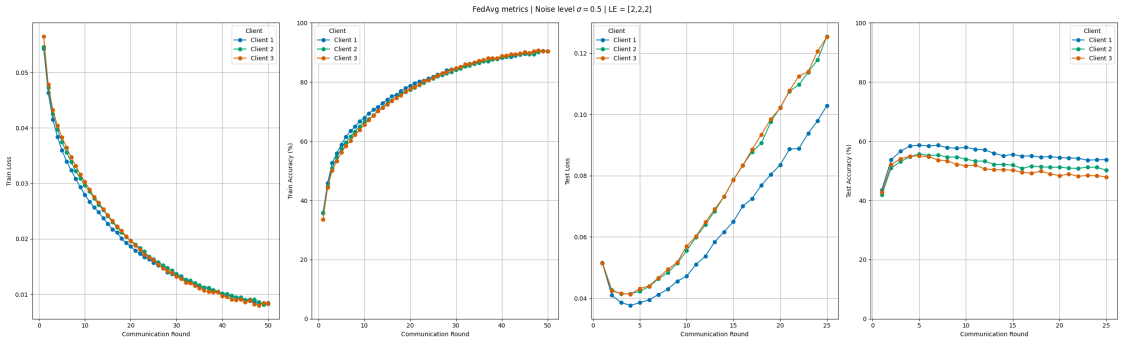

Figure 2.12: Local metrics for 3 clients in 50 communication rounds using FedAvg with a Non-IID setting over the CIFAR10 dataset. Feature distribution skew, noise level $\sigma = 0.5$

From Figure 2.12 we can see that although the clients seem to converge, the model's performance is worst in comparison with a IID setting.
Even though in a real scenario there wouldn't be a global test dataset, a comparison between different data partition settings in addition to a centralized training scenario will be tested with the union of all clients' datasets. This will serve to see how good the global model performs in comparison with a centralized model.

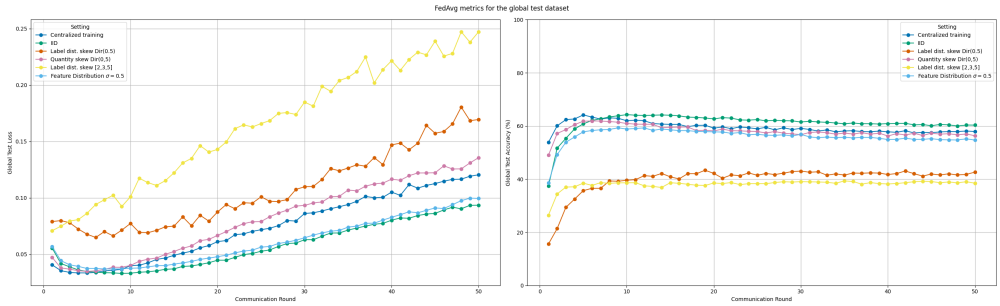

Figure 2.13: Test metric for different data distribution settings using (theoretical) global dataset D compared with a centralized setting.

As it can be seen from Figure 2.13, the centralized, IID, quantity skew and feature distribution skew settings perform quite similar, being both label distribution skew settings the ones that degrades the model the most. From Algorithm 1, each client performs multiple local epochs, which reduces the communication rounds. However, these local updates can lead to a worse performance since the local objective functions differ from each other. This makes the algorithm not robust against non-IID data. There are several works that try to tackle this problem and a lot of FedAvg variants have been proposed in order to mitigate the divergence between the local updates. We will review some of the more relevant of them, starting with *FedProx*.

2.5 FedProx

In [25], a generalization of *FedAvg* was proposed where the local function to minimize $F_k(\omega_k)$ was modified adding a proximal term:

$$\min_{\omega_k} h_k(\omega_k, \omega_g^t) = F_k(\omega_k) + \frac{\mu}{2}\|\omega_k - \omega_g^t\|^2, \qquad \mu \geq 0 \tag{2.3}$$

The basic idea, is to restrict the local updates to be closer to the actual global model. In the original proposal, it was stated that a constant number of local epochs per round each time could be not feasible because of the device / system heterogeneity. Since this work is not focused on system heterogeneity, it will not be mentioned the theoretical convergence results.

FedAvg is a particular case when $\mu = 0$, a fixed number of local epochs E is set and all the local solvers are SGD. As μ increases, the function becomes more restrictive meaning that it will take longer to converge. This algorithm doesn't restrict us to any local solver, so it's not necessary to compute the estimation of the gradient using SGD. When the data across clients is IID, a positive μ could decelerate the convergence, some heuristics could be applied such as decreasing μ as long as the loss functions continues to decrease. Following the experiments in [25], the possible values of μ that will be used in this work are selected from the finite set $\{0.001, 0.01, 0.1, 1\}$

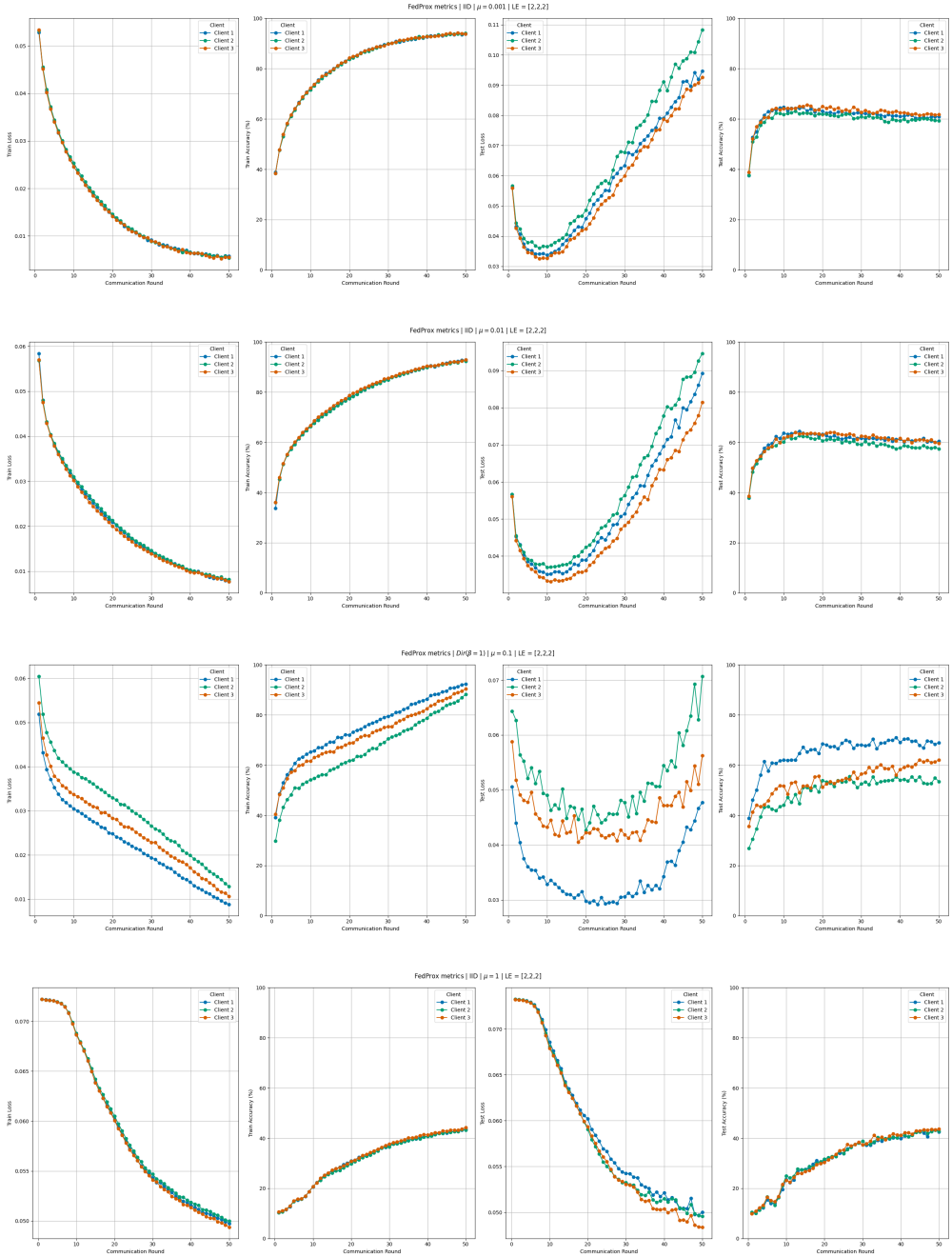

Figure 2.14: Local metrics for 3 clients in 50 communication rounds using FedProx with a IID setting over the CIFAR10 dataset, $\mu \in \{0.001, 0.01, 0.1, 1\}$

Seeing Figure 2.10, according to [25] and [24], A larger μ could decrease the model's convergence even in an IID setting. As of today, only heuristics are available to choose μ; some of them consist of reducing the parameter as training progresses, similar to adjusting the learning rate. The rest of Non-IID setting experiments metrics with FedProx can be seen in Appendix A, for the label distribution skew using $Dir(\boldsymbol{\beta} = (1, 1, 1))$ see Figure 4.8, for a label distribution skew with different labels per party see Figure 4.9, for a label quantity distribution skew using $Dir(\boldsymbol{\beta} = (0.5, 0.5, 0.5))$ see Figure 4.10 and for a feature distribution skew

with noise level $\sigma = 0.5$ see Figure 4.11.
Now only a few cases will be shown where with different values of mu a similar or better result has been obtained compared to FedAvg under the same conditions.

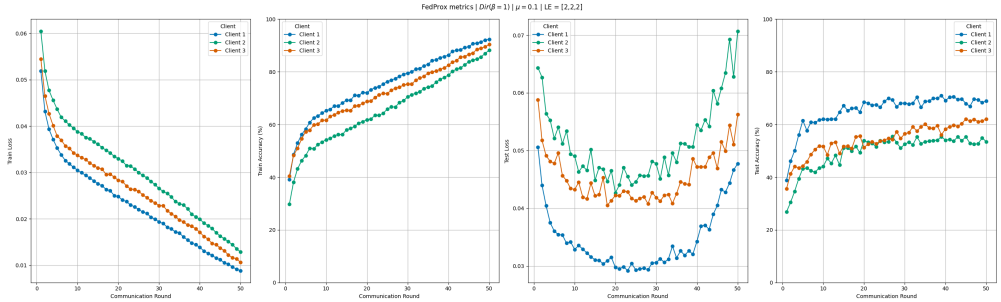

Figure 2.15: Local metrics for FedProx, $\mu = 0.1$, label distribution skew using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (1, 1, 1)$

Comparing Figure 2.9 and Figure 2.15 we can see that the clients' local training improves slightly, specially for client 2 and 3. However, we will compare both models using the global dataset D in order to test if it hasn't been overfitted by a client whose local dataset's cardinal is way bigger that the rest of clients.

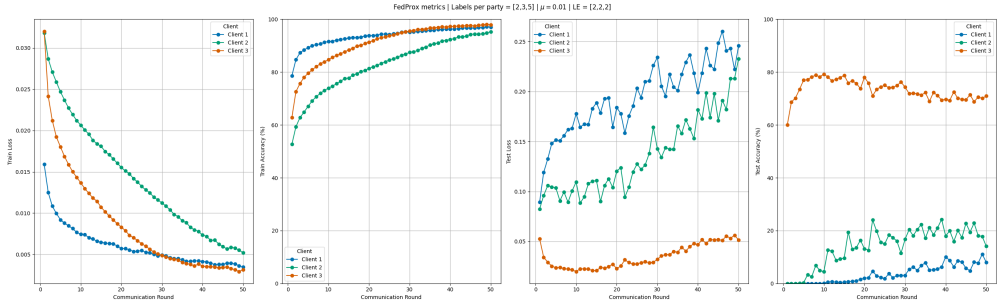

Figure 2.16: Local metrics for FedProx, $\mu = 0.01$, label distribution skew with a disjoint partition of $[2, 3, 5]$.

As we see from Figure 2.16, a better performance has been achieved compared with the same setting using FedAvg, although the improvement is modest and mainly for client 1, whose previous accuracy was near 0 since only holds 2 labels.

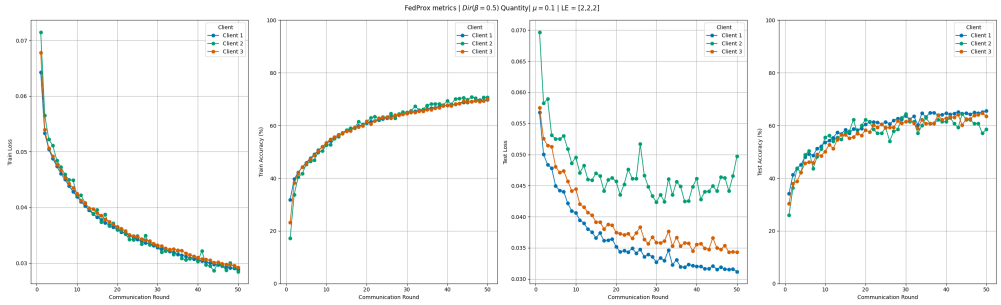

Figure 2.17: Local metrics for FedProx, $\mu = 0.1$, label quantity skew using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$.

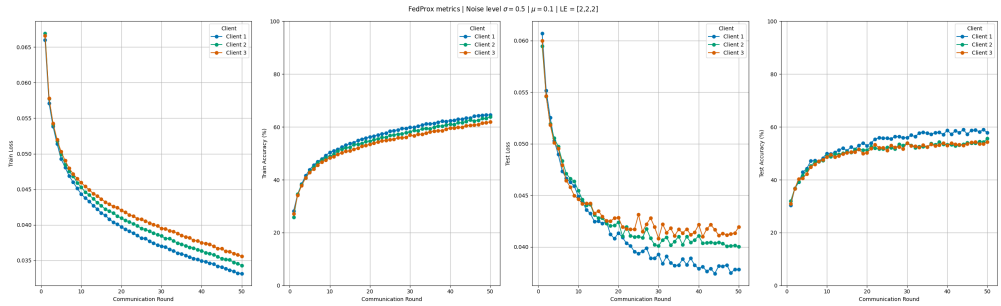

Figure 2.18: Local metrics for FedProx, $\mu = 0.1$, feature distribution skew using a noise level of $\sigma = 0.5$.

In Figure 2.17 all clients have a better test accuracy compared with the FedAvg experiment (in the same setting). For the feature skew, we see that the performance seen in Figure 2.18 is similar to the one in 2.12.

	IID	Label Skew Dir(0.5)	Quantity Skew Dir(0.5)	Labels per Party	Noise $\sigma = 0.5$
FedAvg	**0.6427**	0.4335	0.6192	0.3934	0.5933
FedProx Mu 0.001	0.6425	0.4466	0.6309	0.4034	0.5961
FedProx Mu 0.01	0.6356	**0.4505**	**0.6494**	**0.4435**	**0.5979**
FedProx Mu 0.1	0.6363	0.4347	0.6396	0.4226	0.5906
FedProx Mu 1	0.4443	0.4034	0.4238	0.3437	0.4375

Table 2.1: Top global test accuracy obtained in FedAvg and FedProx under the same settings for different values of μ.

2.6 FedNova

This algorithm was first introduced in [46], in order to study its main contributions we are changing the original notation of *FedAvg*. We have seen that the client updates its model at round t starting with a shared, global model ω_g^t. Then, it applies SGD with its local data for a fixed number of epochs E. We recall that, for each epoch, and for each batch $\mathbf{b} = \{\mathbf{x}, y\} \in \mathcal{B} \subset D^i$, the i-th client performs $\omega_i^t = \omega_i^t - \eta \nabla l_i(\omega_i^t, \mathbf{b})$. After the round, the client will have a locally updated model ω_i^{t+1}. After all the clients have updated their models, it aggregates them following the formula: $\omega_g^{t+1} = \sum_{k \in S_t} \frac{|D^k|}{\sum_{k \in S_t} |D^k|} w_k^{t+1}$. We can now consider the difference between the global client's starting model ω_g^t and its locally updated model ω_i^{t+1}, we will denote $\Delta\omega_i^t := \omega_g^t - \omega_i^{t+1}$, the difference between the global model at time t and the local model of client i at time $t+1$. The server would then collect $\Delta\omega_k^t \ \forall k \in \{1, \ldots, N\}$ and update the global model. For example, in *FedAvg*: $\omega_g^{t+1} = \omega_g^t + \sum_{k \in S_t} \alpha_k \Delta\omega_k^t = \omega_g^t - \sum_{k \in S_t} \alpha_k \eta \sum_{j=1}^{\tau_k} g_k(\omega_k^{t,j})$ where $\alpha_k = \frac{|D^k|}{\sum_{k \in S_t} |D^k|}$, $\tau_k = \lfloor \frac{E_k |D^k|}{B} \rfloor$ with B being the mini-batch size and E_k the local number of epochs of client k, $\omega_k^{t,j}$ is the client k's model after the j-th SGD update at time t, η is the learning rate (same for all clients) and g_k is the stochastic gradient over a mini-batch $\mathbf{b} \in \mathcal{B}$.
As stated in [46], different E_k across clients and rounds means that *FedAvg* algorithm converges to a surrogate objective instead of $F(\omega)$ from (2.1). This algorithm considers that different parties compute a different number of local steps (τ_k) because of computation constraints or different sizes of local datasets. When $\tau_k > \tau_{k'}$, client k will have a major influence over the global update rule compared to client k'. That's why the authors proposed to normalize and scale the local updates according to the number of local steps.

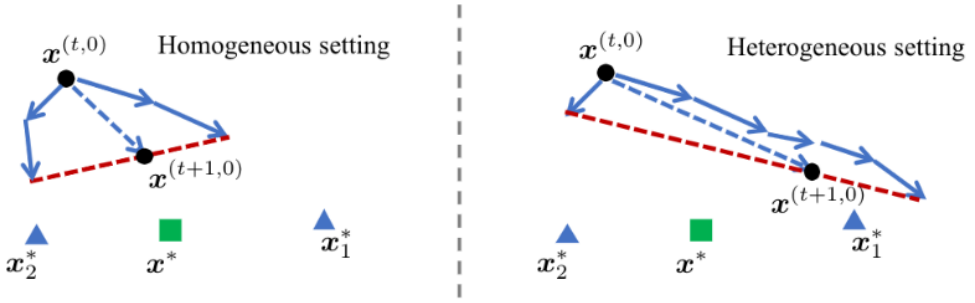

Figure 2.19: Model updates in the parameter space. Green squares and blue triangles denote the minima of global and local objectives, respectively. Source: [46]

Algorithm 2 FedNova

Input: local datasets $D^i \; \forall i \in \{1, \ldots, N\}$, number of parties N, number of communication rounds T, number of local epochs E, learning rate η, local mini-batch size B.
Output: global model ω_g^T.

procedure SERVER EXECUTION
  Initialize ω_g^0
  for round $t = 1, \ldots, T$ **do**
    S_t (Selection of clients)
    for client $k \in S_t$ **in parallel do**
      $\Delta\omega_k^t, \tau_k \leftarrow ClientUpdate(k, \omega_g^t)$
    $n \leftarrow \sum_{k \in S_t} |D^k|$
    $\omega_g^{t+1} \leftarrow \omega_g^t - \eta \frac{\sum_{k \in S_t} |D^k| \tau_k}{n} \sum_{k \in S_t} \frac{|D^k| \Delta\omega_k^t}{\tau_k n}$
procedure $ClientUpdate(k, \omega_g^t)$
  $\omega_k^t \leftarrow \omega_g^t$
  $\tau_k \leftarrow 0$
  $\mathcal{B} \leftarrow$ Batches of D^k of size B
  for local epoch $i = 1, \ldots, E$ **do**
    for batch $\mathbf{b} \in \mathcal{B}$ **do**
      $\omega_k^t \leftarrow \omega_k^t - \eta \nabla l(\omega_k^t; \mathbf{b})$
      $\tau_k \leftarrow \tau_k + 1$
  $\Delta\omega_k^t \leftarrow \omega_g^t - \omega_k^t$
  return $\Delta\omega_k^t$, τ_k to the server.

For the next experiments, the same data distribution and initial weights will be used for both FedAvg and FedNova, starting with an IID setting:

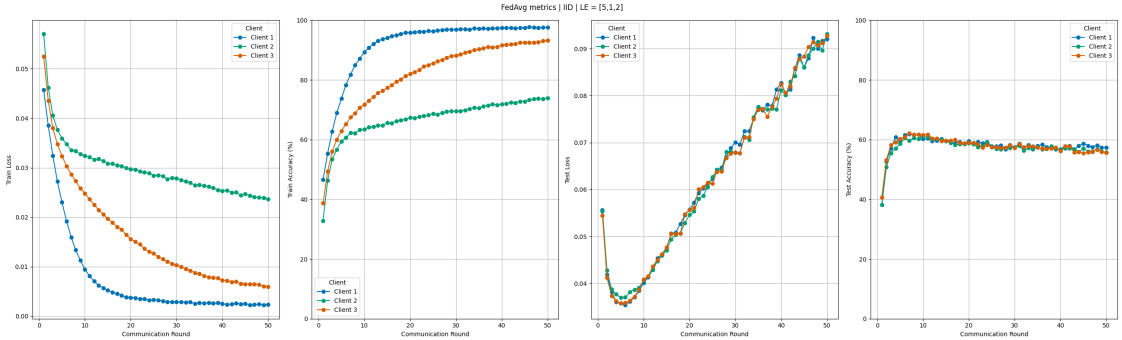

Figure 2.20: Local metrics for FedAvg on a IID setting with local epochs [5,1,2].

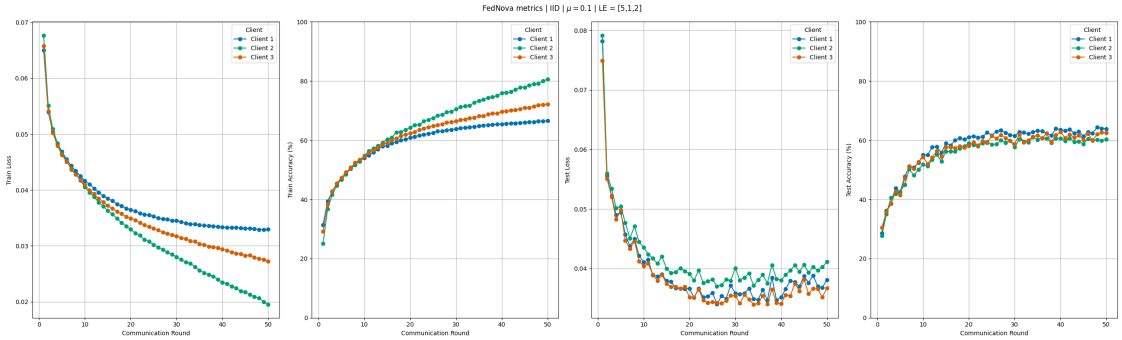

Figure 2.21: Local metrics for FedNova using proximal term with $\mu = 0.1$ on a IID setting and local epochs [5,1,2].

We have used FedNova with a local loss modified as explained with FedProx, since both approach are compatible. We see a slightly improvement with FedNova compared to FedAvg in a IID setting. Note that the first client performs 5 local epochs, the second client only 1 epoch and the third client 2 epochs (assuming some kind of computational or communication constraint). However, for a label skew setting using the Dirichlet distribution, Figures 2.22 and 2.23, the client with more epochs (client 1) got lower test accuracy compared to the others clients, since as the client's data is scarce (see Figure 2.4 for each client's distribution). More local epochs doesn't mean that the client will perform better on its own local data, since the aggregation step changes all its model weights favoring the clients with more data.

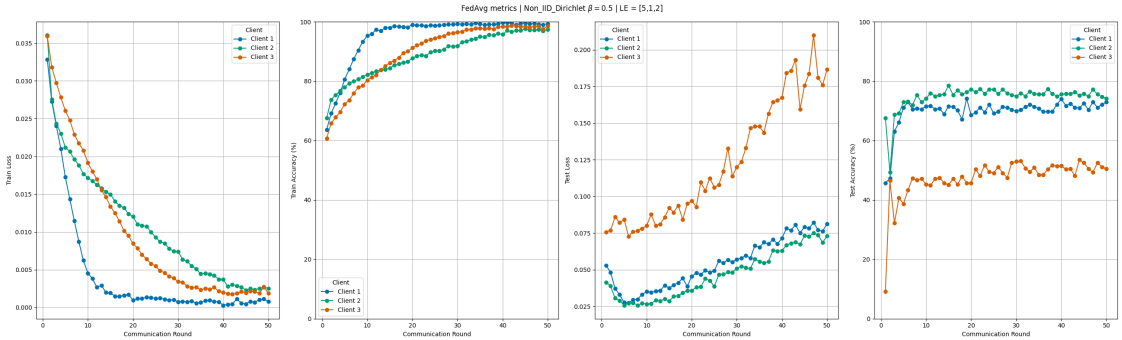

Figure 2.22: Local metrics for FedAvg on a Non-IID label skew using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$ setting with local epochs [5,1,2].

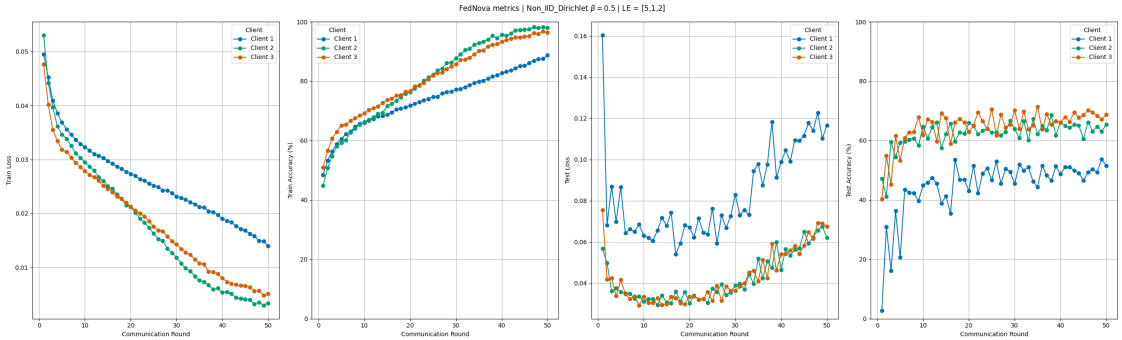

Figure 2.23: Local metrics for FedNova using proximal term with $\mu = 0.1$ on a Non-IID setting label skew using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$ and local epochs [5,1,2].

We see in this case that FedNova doesn't perform better than FedAvg, although we will compare the model's accuracy later with the global dataset. This is something that some articles have already commented: FedNova doesn't perform better than other approaches or even FedAvg in some Non-IID scenarios [24]. In fact, in the next experiment we can see that using FedNova performs worse than FedAvg for both clients 1 and 3, where client 1 holds two labels and client 3 holds 3 labels. From the previous expriment we have seen that this setting is the one that hurts the model accuracy the most.

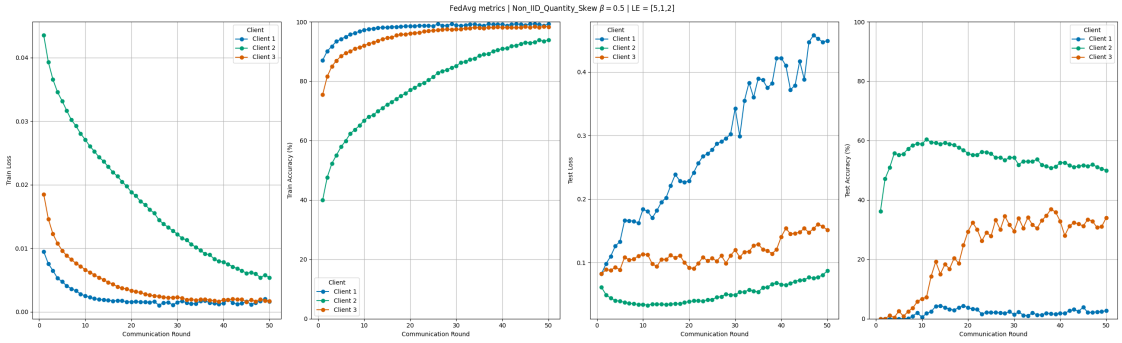

Figure 2.24: Local metrics for FedAvg on a Non-IID setting Quantity-based (labels per party = [2,5,3]) and local epochs [5,1,2].

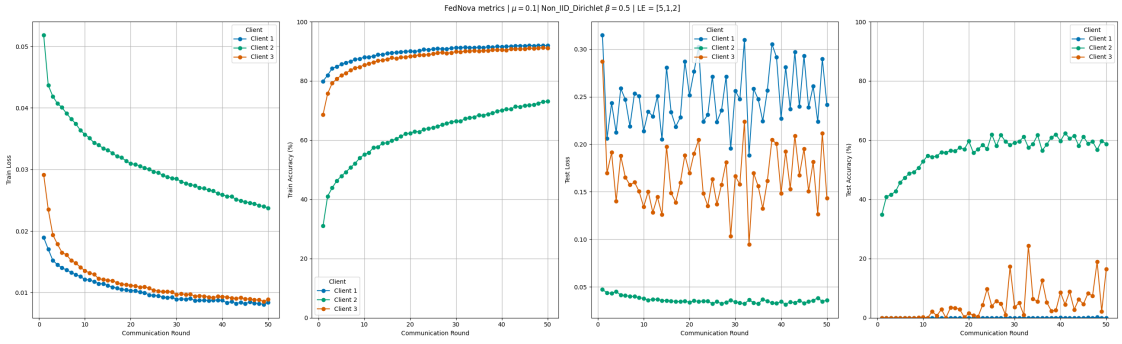

Figure 2.25: Local metrics for FedNova using a proximal term with $\mu = 0.1$ on a Non-IID setting Quantity-based (labels per party = [2,5,3]) and local epochs [5,1,2].

In Figures 2.26 and 2.27, FedNova has slightly better results for clients 1 and 2 but client 3 doesn't converge at all, making FedAvg a more stable approach in this case.

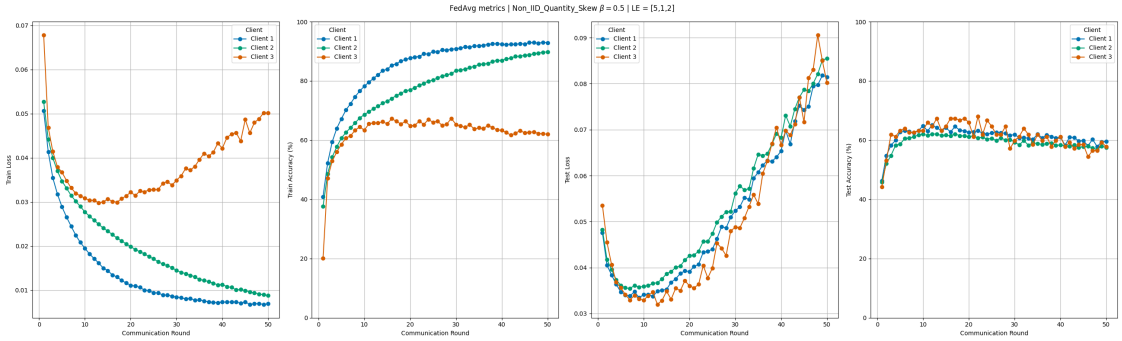

Figure 2.26: Local metrics for FedAvg on a Non-IID Quantity Skew setting using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$ and local epochs [5,1,2].

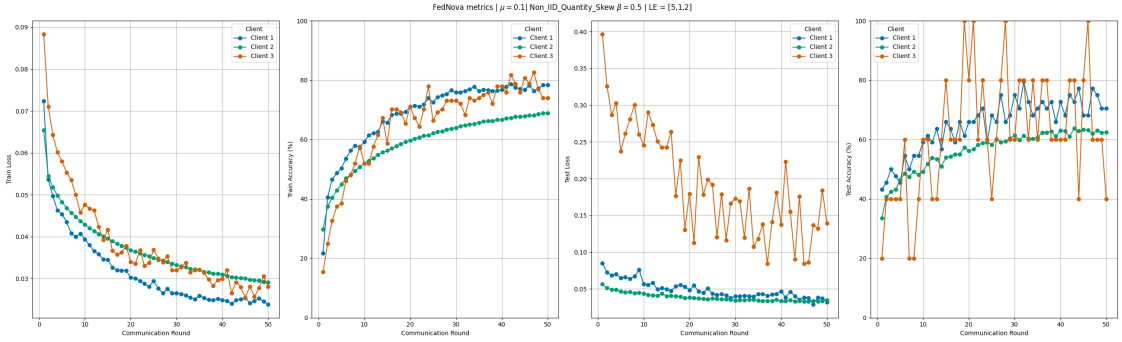

Figure 2.27: Local metrics for FedNova using a proximal term with $\mu = 0.1$ on a Non-IID Quantity Skew setting using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$ and local epochs [5,1,2].

Finally, for a feature skew experiment, we can see from Figures 2.28 and 2.29 that both approaches are similar, with FedNova having a slightly better test accuracy in all clients.

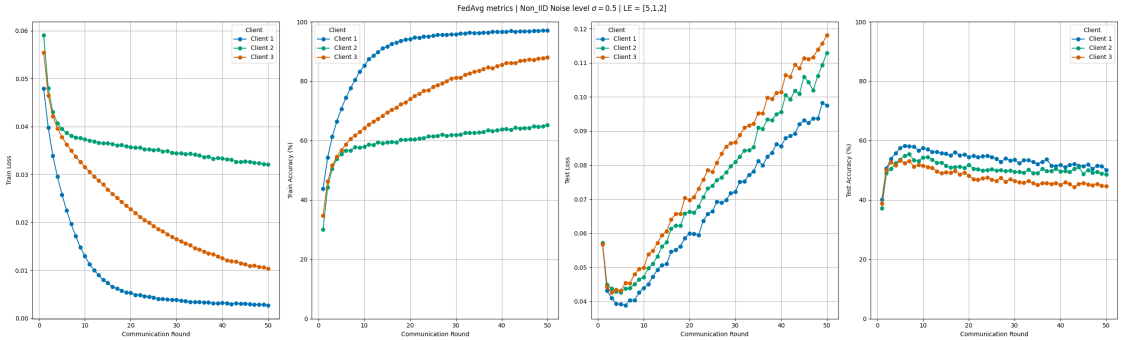

Figure 2.28: Local metrics for FedAvg on a Non-IID Feature Skew setting with noise level $\sigma = 0.5$ and local epochs [5,1,2].

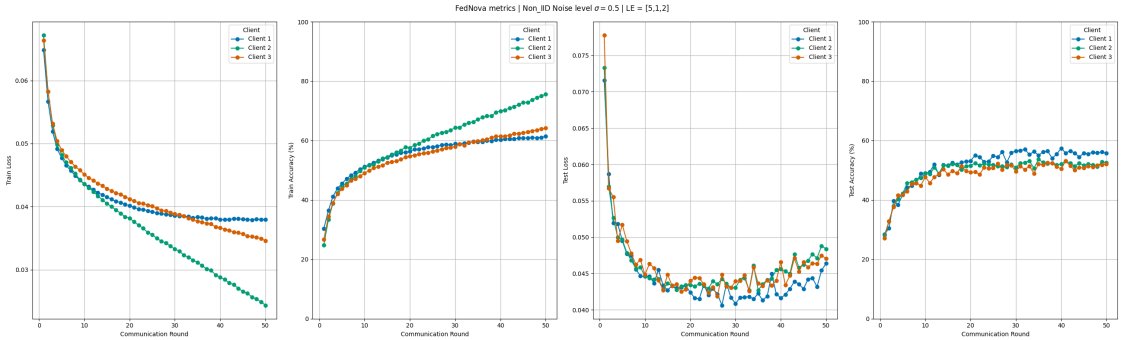

Figure 2.29: Local metrics for FedNova using a proximal term with $\mu = 0.1$ on a Non-IID Feature Skew setting with noise level $\sigma = 0.5$ and local epochs [5,1,2].

	IID	Label Skew Dir(0.5)	Quantity Skew Dir(0.5)	Labels per Party	Noise $\sigma = 0.5$
FedAvg	0.616	**0.4328**	0.6246	**0.3818**	**0.5834**
FedNova	**0.6231**	0.427	**0.6456**	0.3552	0.5749

Table 2.2: Top test accuracy obtained in FedAvg and FedNova under the same settings.

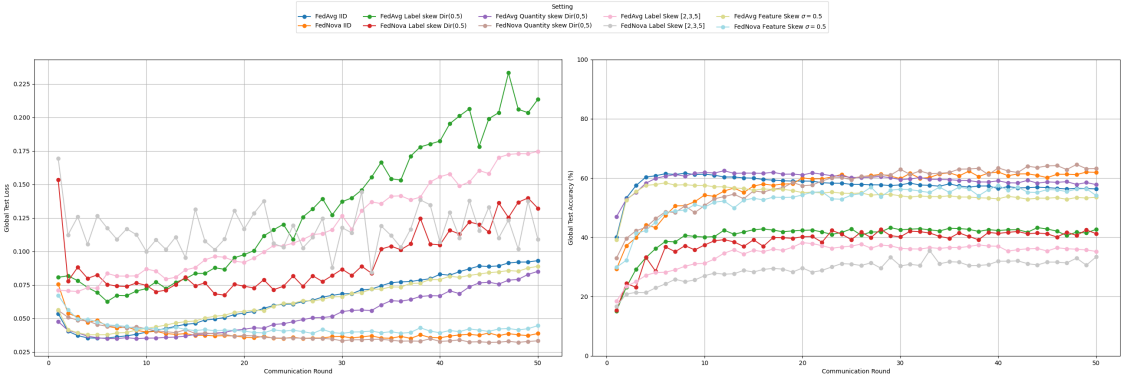

Figure 2.30: Accuracy comparison between FedAvg and FedNova on the global test dataset.

As shown in Figure 2.30, FedNova cannot mitigate the effect of Label Skew either, being, as with FedProx, the most detrimental setting for the model.

2.7 Other algorithms to train with Non-IID data

Among several algorithms that can be used for training a FL model under a non-IID setting, we mention some of the most performant:

SCAFFOLD:

With data heterogeneity, *FedAvg* suffers from client-drift, since each local updates it's minimizing its own local objective. SCAFFOLD tries to solve this introducing control variables for the server c and the clients $c_k \ \forall k \in \{1, \ldots, N\}$.

Algorithm 3 SCAFFOLD

Input: local datasets $D^i\ \forall i \in \{1,\dots,N\}$, number of parties N, number of communication rounds T, number of local epochs E, learning rate η, local mini-batch size B.
Output: global model ω_g^T.

1: **procedure** SERVER EXECUTION
2: Initialize ω_g^0
3: $c^t \leftarrow \mathbf{0}$, $c_k \leftarrow \mathbf{0}$
4: **for** round $t = 1,\dots,T$ **do**
5: S_t (Selection of clients)
6: **for** client $k \in S_t$ **in parallel do**
7: $\Delta\omega_k^t, \Delta c_k \leftarrow ClientUpdate(k, \omega_g^t, c^t)$
8: $n \leftarrow \sum_{k\in S_t} |D^k|$
9: $\omega_g^{t+1} \leftarrow \omega_g^t - \eta \sum_{k\in S_t} \frac{|D^k|}{n} \Delta\omega_k^t$
10: $c^{t+1} \leftarrow c^t + \frac{1}{N}\sum_{k\in S_t} \Delta c_k$
11: **procedure** $ClientUpdate(k, \omega_g^t, c_k)$
12: $\omega_k^t \leftarrow \omega_g^t$
13: $\tau_k \leftarrow 0$
14: $\mathcal{B} \leftarrow$ Batches of D^k of size B
15: **for** local epoch $i = 1,\dots,E$ **do**
16: **for** batch $\mathbf{b} \in \mathcal{B}$ **do**
17: $\omega_k^t \leftarrow \omega_k^t - \eta(\nabla l(\omega_k^t; \mathbf{b}) - c_k + c)$
18: $\tau_k \leftarrow \tau_k + 1$
19: $\Delta\omega_k^t \leftarrow \omega_g^t - \omega_k^t$
20: $c_k^+ \leftarrow$ (i) $\nabla l(\omega_g^t)$ or (ii) $c_k - c + \frac{1}{\tau_k \eta}(\omega_g^t - \omega_k^t)$
21: $\Delta c \leftarrow c_k^+ - c_k$
22: $c_k = c_i^+$
23: return $\Delta\omega_k^t$, Δc to the server.

From the SCAFFOLD's algorithm, it can be noted that the clients maintain the state of c_k and the server maintain c, which are initialized at 0. In comparison with the previous algorithms, here the local update formula is: $\omega_k^t = \omega_k^t - \eta(\nabla l(\omega_k^t, \mathbf{b}) - c_k + c)$. Intuitively, the local gradient $\nabla l(\omega_k^t, \mathbf{b})$ moves towards the local optimum for client k, the correction $c - c_k$ ensures that the update change its direction towards the global optimum. As we can see, in line 20, there are two options for computing c_i^+: $\nabla l(\omega_g^t)$ or $c_k - c + \frac{1}{\tau_k\eta}(\omega_g^t - \omega_k^t)$, that is, computing the gradient of the local data at the global model or reusing the previously computed gradients. The first approach is more stable but the second one has lower computation cost.

FedNAG [51]: Using Nesterov Accelerated Gradient as the optimizer has proved to be a more efficient paradigm that training with FedAvg or FedSGD. For a basic description of this method see Section 1.4 and Equation 1.5. In the FL context, this stochastic gradient descent variant is modified as follow:
Let $\boldsymbol{w}_i(t)$ and $\boldsymbol{v}_i(t)$ denote the model parameter and momentum parameter in client i at tth iteration. Each τ iterations (local epochs) leads to a global aggregation, $t = k\tau$, $k = 1, 2, ...$ (the original paper assumes that all clients perform the same number of local epochs). In each iteration, client i performs:

$$\begin{aligned}\boldsymbol{v}_i(t) &= \gamma \boldsymbol{v}_i(t-1) - \eta \nabla F_i(\boldsymbol{w}_i(t-1)) \\ \boldsymbol{w}_i(t) &= \boldsymbol{w}_i(t-1) - \gamma \boldsymbol{v}_i(t-1) + (1+\gamma)\boldsymbol{v}_i(t) \\ &= \boldsymbol{w}_i(t-1) + \gamma \boldsymbol{v}_i(t) - \eta \nabla F_i(\boldsymbol{w}_i(t-1))\end{aligned}$$

In the aggregation step, the central sever performs:

$$\boldsymbol{v}(t) = \frac{\sum_i |D^i| \boldsymbol{v}_i(t)}{\sum_j |D^j|}$$
$$\boldsymbol{w}(t) = \frac{\sum_i |D^i| \boldsymbol{w}_i(t)}{\sum_j |D^j|}$$

FedAdagrad and **FedAdam**: Based on Adagrad optimizer explain in Equation 1.6 and ADAM optimizer explained in equation 1.8, [34] proposed an adaptive federated version of these algorithms in order to improve the convergence results of FedAvg, following its notation FedAvg's update can be expressed as:

$$\omega_g^{t+1} = \omega_g^t - \frac{1}{N} \sum_{i=1}^{N} (\omega_g^t - \omega_i^t)$$

Let $\Delta_i^t := \omega_i^t - \omega_g^t$ and $\Delta_t := \frac{1}{N} \sum_{i=1}^{N} \Delta_i^t$. The server update in FedAvg is like applying SGD to $-\Delta_t$ with learning rate $\eta = 1$ (see FedOpt algorithm in [34]). Therefore, the will perform the local training for each epoch k computing the unbiased gradient estimate $g_{i,k}^t$ of $\nabla F_i(\omega_{i,k}^t; x)$ and updating the weights $\omega_{i,k+1}^t = \omega_{i,k}^t - \eta_l g_{i,k}^t$. After computing all local epochs of training, each client computes $\Delta_i^t = \omega_i^t - \omega_g^t$. Then, the server will get $\Delta_t = \frac{1}{N} \sum_{i=1}^{N} \Delta_i^t$ and the first moment estimate (with the decay parameters $\beta_1, \beta_2 \in [0, 1)$) $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta_t$. Then, adapting the optimizers from Section 1.4, FedAdagrad computes the second moment estimate $v_t = v_{t-1} + \Delta_t^2$ and FedAdam $v_t = \beta_2 v_{t-1} + (1 - \beta_2)\Delta_t^2$. Finally, the global model is updated $\omega_g^{t+1} = \omega_g^t + \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$.

There are numerous algorithms aimed at mitigating the effects of statistical heterogeneity in federated training. Some, like those mentioned, are based on optimizers. Others select the clients that can collaborate in training by clustering them according to the similarity of their local data. In some cases, it is proposed to pre-train the global model with a public dataset, which may not be feasible in real-world scenarios unless synthetic data is used. Personalization methods such as local fine-tuning or knowledge distillation have also been proposed. In [57], different methods currently available to prevent model degradation due to non-IID data can be found.

Chapter 3

Machine Learning on Vertical-Partitioned Dataset

Vertical partitioning involves splitting a dataset based on features, where each data holder possesses a subset of the features for the same set of individuals. This differs from horizontal partitioning, where different entities hold data for different individuals. The goal is to enable collaborative learning without compromising the privacy of the individuals whose data is being used.
For example, consider two hospitals aiming to improve diagnostic models by pooling their patient data. Each hospital has different pieces of patient information (e.g., one has genetic data while the other has imaging data). Vertical partitioning allows these hospitals to build a comprehensive model using their combined data without sharing their raw datasets.

Private Set Intersection: Private Set Intersection (PSI) is a cryptographic technique that enables two parties to compute the intersection of their datasets without revealing any non-intersecting elements. This ensures that only the common elements are identified and used for further analysis.
PSI is crucial in vertical partitioning because it allows different entities to match records corresponding to the same individual without sharing sensitive information. For instance, if a pharmaceutical company wants to collaborate with a hospital to identify patients who have received a particular treatment and also have specific genetic markers, PSI can be employed. This method ensures that neither party learns more than necessary about the other's dataset.
There are several variations of PSI, each offering different trade-offs between computational efficiency and security. Basic PSI protocols often rely on cryptographic primitives such as hashing, oblivious transfer, and homomorphic encryption. Advanced PSI protocols improve efficiency and scalability, making them suitable for large datasets often encountered in real-world applications.

Split Neural Network: Split Neural Networks (SplitNN) provide another approach to vertical partitioning by distributing the training of a neural network across multiple parties. In SplitNN, each party trains a portion of the network up to a certain layer, then passes the intermediate results (activations) to the next party. This process continues until the final output layer is reached.
The key advantage of SplitNN is that it keeps the raw data within each party's domain, only sharing the intermediate activations. This greatly enhances data privacy while still allowing for collaborative model training. For example, in a healthcare scenario, one hospital could train the initial layers of a neural network on imaging data, pass the intermediate activations to another hospital, which then continues training on genetic data.
SplitNN also offers the flexibility to handle different types of data and model architectures, making it suitable for various machine learning tasks. Additionally, it supports both supervised and unsupervised learning, further broadening its applicability.

3.1 PSI Protocol and implementation

One widely used application of MPC is **Private Set Intersection** (**PSI**), which allows to compute the common elements between datasets' parties (for simplicity, we will focus on two parties). Given a party P_i with a dataset D^i, $i = 1, 2$, we want to compute $D^1(j) \cap D^2(j)$ being $D^i(j)$ the j-th column (set) from the i-th party's dataset. For example, if D consist in a single list of phone numbers, $D^1 \cap D^2$ would be the common phone contacts between party 1 and party 2. Another example could be that the datasets D^1 and D^2 have the fields:

- $D^1(1) \coloneqq$ National Identity Number
- $D^1(2) \coloneqq$ Name
- $D^1(3) \coloneqq$ Surname
- $D^1(4) \coloneqq$ Annual Salary ($)

- $D^2(1) \coloneqq$ National Identity Number
- $D^2(2) \coloneqq$ Age
- $D^2(3) \coloneqq$ Genre

If we would like to know the mean annual salary by genre, first we would need to *merge* both datasets, mapping the National Identity Number. With PSI, we could compute $D^1(1) \cap D^2(1)$, this gives us the NIN of the people with complete data, i.e. the people whose personal data are in both datasets, so it is known both the annual salary and the genre. With that intersection, we could get two subsets from D^1 and D^2 and then use another MPC protocol (in this case, we are interested in the operation *GroupBy*).

As an informal introduction of the problem, we will develop the protocol studied in [2]. We define two sets: $V_A = \{v_1, \cdots, v_n\}$ and $V_B = \{v'_1, \cdots, v'_m\}$, $n, m \in \mathbb{N}$ which belong to node Alice and Bob resp. For this protocol, it is needed a commutative encryption $\mathcal{F}$: a computable, in polynomial time, function $f\colon Key(\mathcal{F}) \times Dom(\mathcal{F}) \to Dom(\mathcal{F})$, defined on a finite domain, that satisfies the properties:

- Commutativity: $\forall e, e' \in Key(\mathcal{F})$, $f_e \circ f_{e'} = f_{e'} \circ f_e$
- $f_e\colon Dom(\mathcal{F}) \to Dom(\mathcal{F})$ is a bijection $\forall e \in Key(\mathcal{F})$
- f_e^{-1} is also computable in polynomial time given e.
- Given a value x and its encryption $f_e(x)$, for a new value y we cannot distinguish between $f_e(y)$ and a random value z in polynomial time.

Let $Dom(F)$ be all quadratic residues modulo p, where p is a "safe" prime number (both p and $q = \frac{p-1}{2}$ are primes). Let $Key(\mathcal{F})$ be $\{1, 2, ..., q-1\}$. Then, the power function $f_e(x) \equiv x^e mod(p)$ is a commutative encryption.
In order to map the elements from X and Y (which are assumed to be set of unique elements, such as NIN) into $Dom(\mathcal{F})$, we need a hash function $h\colon V \to Dom(\mathcal{F})$, $V_A \cup V_B \subset V$, that can be considered computed by a random oracle: for each $v \in V$, an independent random $x \in_r Dom(\mathcal{F})$[1] is chosen for $x = h(v)$. In order to avoid a collision, $|Dom(\mathcal{F}) >> |X \cup Y|$. Let $N = |Dom(\mathcal{F})|$, in the random oracle model the probability that n hash values have at least one collision is:

$$P(hash\ collision) = 1 - \prod_{i=1}^{n-1} \frac{N-i}{N} \approx 1 - \exp\big(\frac{-n(n-1)}{2N}\big)$$

The intersection protocol described in [2] is:

Note that Algorithm 4 can be modified so all the transmission operations are symmetrical, therefore both Alice and Bob know $V_A \cap V_B$. A toy implementation[2] has been done in Python using gRPC, although this is a very basic implementation and shouldn't be considered for commercial use, since it has no guaranties of security nor privacy.

[1] $\in_r$ means: chosen uniformly at random from

[2] Source code in `https://github.com/JSempereH/TFM/tree/main/src/3-PSI`

Algorithm 4 Private Set Intersection Protocol

1: Both Alice and Bob apply the hash function h to their sets: $X_A = h(V_A)$ and $X_B = h(V_B)$. Each party randomly chooses a secret key: $e_A \in \text{Key}_F$ for Alice and $e_B \in \text{Key}_F$ for Bob.
2: Both parties encrypt their hashed sets: $Y_A = f_{e_A}(X_A) = f_{e_A}(h(V_A))$ and $Y_B = f_{e_B}(X_B) = f_{e_B}(h(V_B))$.
3: Bob sends to Alice his encrypted set $Y_B = f_{e_B}(h(V_B))$, reordered lexicographically.
4: Alice sends to Bob her set $Y_A = f_{e_A}(h(V_A))$, reordered lexicographically.
5: Alice encrypts each $y \in Y_B$ with her key e_A and sends back to Bob pairs $\langle y, f_{e_A}(y)\rangle = \langle f_{e_B}(h(v)), f_{e_A}(f_{e_B}(h(v)))\rangle$.
6: Bob encrypts each $y \in Y_A$ with e_B, obtaining $Z_A = f_{e_B}(f_{e_A}(h(V_A)))$. Also, from pairs $\langle f_{e_B}(h(v)), f_{e_A}(f_{e_B}(h(v)))\rangle$ obtained in Step 4(b) for $v \in V_B$, he creates pairs $\langle v, f_{e_A}(f_{e_B}(h(v)))\rangle$ by replacing $f_{e_B}(h(v))$ with the corresponding v.
7: Bob selects all $v \in V_B$ for which $f_{e_A}(f_{e_B}(h(v))) \in Z_A$; these values form the set $V_A \cap V_B$.

For learning purposes, let's suppose Alice has a dasatet with the following NINs: $12345678A$, $87654321B$, $54321678C$, $67891234D$, and Bob has a dataset with the following NINs: $67891234D$, $22222222Y$, $54321678C$, $11111111X$. Using a non-cryptographic hash function[3] we perform step 1 from Algorithm 4:

- X_A = [6801962795022291115, 14914775733042932484, 6031741069150041195, 7775431530089967261]
- X_B = [7775431530089967261, 15704629945399610882, 6031741069150041195, 8717288097697391268]

Using $p = 9223372036854771239$ and $q = 4611686018427385619$, Alice's secret key $e_A = 2671390701970302872$ and Bob's secret key $e_B = 325637035052474817$, we perform step 2 encrypting $x^e \, mod(p)$ for $x \in X_A, X_B$ and $e \in e_A, e_B$:

- Y_A = [4037654302475096183, 7215131381927850152, 7524770399191986516, 4530009952158919715]
- Y_B = [709144700394714309, 1186382732300108902, 683568481448076414, 3441729271243177027]

Both parties send its sets reordered (denoted by $\hat{Y}$), Bob receives Alice's ordered set and after encrypting, Bob sends back to Alice the set:

- $f_{e_B}(\hat{Y}_A)$ = [5211193962823641457, **7666888661095677275**, 4407230382765344063, **3478216981347847458**]

Symmetrically, Alice receives Bob's ordered set and after encrypting, Alice sends back to Bob the set:

- $f_{e_A}(\hat{Y}_B)$ = [**3478216981347847458**, **7666888661095677275**, 8512757503240690236, 6052532624285366477]

Since we are using a commutative encryption scheme, $f_{e_A} \circ f_{e_B} = f_{e_B} \circ f_{e_A}$, each party can compare the received set w.r.t the one encrypted and, using the same indexes obtained by ordering in step 2, both parties would get that the common NINs are $67891234D$ and $54321678C$. This exact example can be executed in the script **PSI.py** in the source code.

3.2 Split Neural Network

It can be the case where different parties have vertical partitioned data where the label of interesting (in a classification task) is only present in one of them. In this case, the features may be in one party and the labels in the other. We could use PSI (see Section 3.1) in order to get the common subset and align the datasets. However, we can't use a distributed Machine Learning like the one explained in Section 2, since the datasets are different across parties, therefore there isn't a common model architecture and we can't simply average the gradients. What we could do is *split* a neural network, with its first layers being adequate for the first party and last layers being appropriate for the second one.

Algorithm 5 Example of SplitNN for 2 parties

1: Alice initializes the weights of its split $F_A = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_n\}$
2: Bob initializes the weights of its split $F_B = \{\boldsymbol{z}_{n+1}, ..., \boldsymbol{z}_N\}$
3: **While** Alice has new data to train on and $epoch < MAX\_EPOCHS$ **do**
4: Alice uses standard forward propagation on data and sends $\boldsymbol{z}_n = \psi(\boldsymbol{W}_n \boldsymbol{z}_{l-1} + \boldsymbol{b}_l)$ to Bob.
5: Bob uses the received $\boldsymbol{z}_n$ as an input to its split F_B and get the output $\boldsymbol{z}_N$.
6: Bob computes the gradients of F_B using Backpropagation and updates its weights.
7: Bob sends the gradient of $\boldsymbol{z}_n$ to Alice.
8: Alice backpropagates gradients received updating the weights of F_A.

Let's suppose Alice has the features $\boldsymbol{x}$ and Bob the labels y, and the datasets are already aligned. Following the notation in Section 1.3, we could summarize the steps as:

Unlike FL, the computation and communication cost is very low in SplitNN. Moreover, aside from the transmission of gradients, the neural network architecture would be identical to that of a centralized model, thus achieving similar performance. For more information on this type of framework and a generalization for N parties see [19]. A basic implementation can be found in the source code of this work.[4]. For Alice, a multilayer perceptron with a first layer of 784 units, a hidden layer of 128 and another one of 640, all fully connected layers with ReLU activation functions (see Section 1.3 for more information). For Bob, a layer of 640 units with ReLU and an output layer of 10 units with LogSoftMax activation function (for a classification task). It has been trained with MNIST dataset (see [10] for more information about this dataset) during 10 epochs following Algorithm 5.

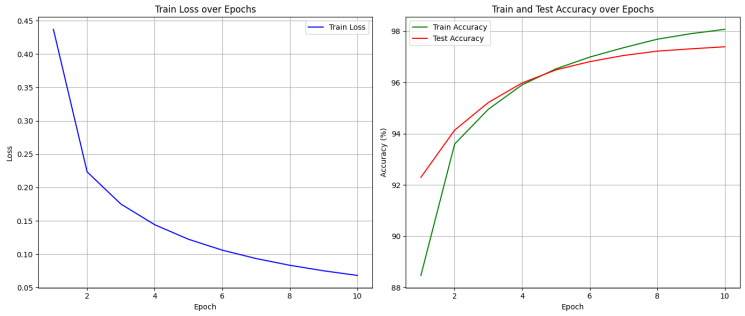

Figure 3.1: Example of SplitNN using MNIST dataset. The training performs the same as a centralized model.

As it can be seen from Figure 3.1, there isn't any model degradation, with just 10 epochs the accuracy is close to 100%. The implementation of Algorithm 5 is quite naive, since there is no serialization of the gradients and everything is computed locally. In a real scenario, different situations would need to be considered, such as the participation of multiple nodes, the (secure) transmission of gradients, model performance monitoring, etc. The provided code is simply a very basic version of how the SplitNN algorithm could be implemented for two computation nodes.

Of course, there are many more algorithms for training models on vertically partitioned data. A fairly comprehensive list of VFL algorithms can be found in [50]. Most techniques require the use of homomorphic encryption or MPC. These technologies are beyond the scope of this work, but I recommend checking [15], [12] and [13] for an introduction.

[3] We don't need a secure hash function, since only serves to map strings to $Dom(\mathcal{F})$. In the implementation, we use SHA1 [39]

[4] `https://github.com/JSempereH/TFM/tree/main/src/3-SplitNN`

Chapter 4

Advanced strategies for Privacy Preserving Machine Learning

This final chapter delves into advanced strategies for enhancing privacy in machine learning, spotlighting two pivotal concepts: Differential Privacy and Generative Models. Each of these strategies offers unique mechanisms for ensuring that sensitive data remains secure and confidential throughout the machine learning lifecycle.

Differential Privacy provides a robust mathematical framework for ensuring that the output of a machine learning model does not compromise the privacy of individual data points. By incorporating controlled noise into the data or the learning process, differential privacy aims to make it difficult for adversaries to infer any specific information about an individual, even if they have access to the model's output.

Generative Models, like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), offer innovative approaches to creating synthetic data that can be used for training machine learning models. These synthetic datasets can mimic the statistical properties of real data without exposing any sensitive information, providing a valuable tool for data sharing and collaborative research without compromising privacy.

This chapter will provide a comprehensive overview of these advanced privacy-enhancing strategies, discussing their principles, implementations, and the challenges associated with their deployment. By integrating these techniques, we can build more secure and privacy-preserving machine learning systems that respect and protect user data in an increasingly data-driven world.

4.1 Differential Privacy

Differential Privacy (DP) is a formal framework designed to ensure the privacy of individuals in a dataset. It operates on the principle of introducing randomness to the data or the analysis process to mask the presence or absence of any single individual. This makes it difficult for adversaries to glean specific information about any individual, even if they have extensive background knowledge. Differential Privacy has become a cornerstone of privacy-preserving data analysis, used by major organizations like Google, Apple, and the US Census Bureau.

4.1.1 Principles of Differential Privacy

The core idea of Differential Privacy is to provide guarantees that the output of a function (algorithm) will be roughly the same whether or not any single individual's data is included in the input dataset. This is typically achieved by adding carefully calibrated noise to the data or to the function's output. The degree of noise is governed by two parameters: ϵ (epsilon) and δ (delta).

- ϵ **(Epsilon):** Represents the privacy loss parameter. Smaller values of ϵ indicate stronger privacy guarantees, but potentially less accurate results.
- δ **(Delta):** Represents the probability that the privacy guarantee does not hold. A smaller δ value indicates a higher level of confidence in the privacy guarantee.

Mathematically, a randomized algorithm A is (ϵ, δ)-differentially private[1][11] if for any two datasets D and D' that differ by only one element, and for any possible output S of the algorithm:

$$Pr[A(D) = S] \leq e^{\epsilon} \times Pr[A(D') = S] + \delta$$

This definition ensures that the inclusion or exclusion of a single individual's data does not significantly affect the outcome, thereby preserving privacy. Therefore, this mathematical framework guarantees that anyone seeing the result of a differentially private analysis will make the same inference about any individual's private information, whether that individual's information in included or not [47]. Note that $\epsilon = \delta = 0$ is equivalent to absolute privacy: the algorithm A is independent of the data. Therefore, there is no accuracy in the queries and this algorithm would be useless for a data analysis.

4.1.2 Private Aggregation of Teacher Ensembles

In [14], it is shown that a model inversion attack that exploits confidence values revealed along with prediction would lead to an unexpected privacy issues. In order to avoid this there are several techniques, we will focus on one algorithm based in differential privacy that provides strong privacy guarantees for training data. For this section, it will be explained the algorithm Private Aggregation of Teacher Ensembles (PATE) [32], which trains multiple models (teachers) on disjoint datasets (local data from parties, as explained in Section 2). These models remain local (like in a federated learning setting) and act as teachers for a *student* model.

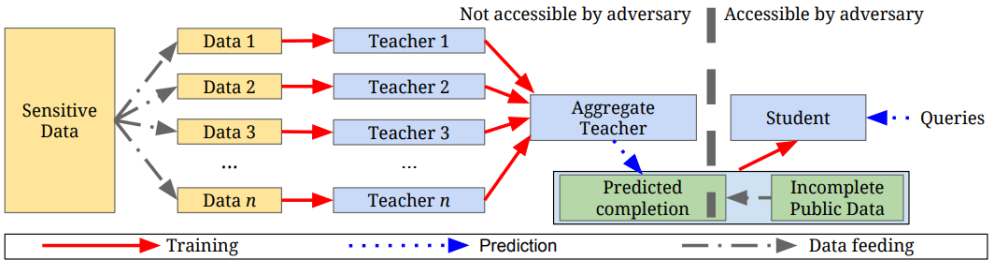

Figure 4.1: Overview of PATE: an ensemble of teachers is trained on disjoint subsets of the sensitive data and a student model is trained on public data labeled using the ensemble. Source: [32]

Suppose we have a public dataset $\{\mathbf{x}, y\}$ where $\mathbf{x}$ denotes the features and y the set of labels. We partition the data in n disjoint sets following the IID hypothesis D^i, $i = 1, ..., n$ and each teacher model trains separately on each set, obtaining n classifiers f_i. We then deploy them as an ensemble making predictions on unseen inputs $\mathbf{x}$ by aggregating into a single prediction the outputs $\{f_i(\mathbf{x})\}_{i=1}^{n}$. The privacy guarantees of the teacher ensemble stems from the aggregation step: let m be the number of classes in the classification task, the label count for a given class $j \in [m]$ and $n_j(\mathbf{x}) = |\{i : i \in [n], f_i(\mathbf{x}) = j\}|$ the number of teaches that assigned class j to input $\mathbf{x}$. Under a majority vote where the label with the largest count is the final prediction, the ensemble's decision may depend on a single teacher's vote (if two labels have a vote count differing by at most one). A random noise is added to the vote count n_j: $f(x) = \arg\max_j \{n_j(\mathbf{x} + Lap(\frac{1}{\gamma}))\}$, where γ is a privacy parameter (not exactly ϵ, although [32] explains its relationship in Theorem 2), and $Lap(\gamma)$ is the Laplacian distribution with location 0 and scale γ. a larger γ leads to a stronger privacy guarantee but in a

[1]Sometimes, when $\delta = 0$ is referred as *pure differentially private*, and otherwise as *approximate differentially private*

degradation of the accuracy, since the maximum f can differ from the true majority. From the Fundamental Law of Information Recovery [11], the noise required would increase as we make more predictions, besides the teachers have been trained on raw data, being susceptible of a membership inference attack [40]. For this reason, another model (the student) is trained using a fixed number of labels predicted by the teacher ensemble. The student model is trained on unlabeled data, some of which it's labeled using the aggregation mechanism. Unlabeled inputs are used in unsupervised learning to estimate a good prior for the distribution while the labeled inputs are then used for supervised learning. Over time, the student learns to generalize from these noisy, aggregated answers, resulting in a model that performs well while preserving the privacy of the original training data.

This algorithm has some limitations: it's computationally expensive since it needs to train multiple teacher models, each on a different subset of data. In the aggregation step, it needs to get the response from the teachers, therefore is not suitable for real-time applications. Moreover, the number of teacher models and the privacy constants add more hyperparameters to the ML workflow, requiring more tuning to balance the model performance and the privacy. It also needs to distribute the data between the teachers in a IID setting, which means we need to have access to some kind of public data (which may not be the case in most real scenarios).
Although it was one of the first and most known algorithm in the PPML community, there are other algorithms and approaches more suitable for a broader use cases. However, because of its simplicity it's still used, and some variants have been proposed: [27], [20].

4.1.3 Differentially Private SGD

Recall from Section 1.4 that for complex NN $J(\theta; x)$ is usually non-convex and difficult to minimize, and in practice a mini-batch stochastic gradient descent (SGD) or some variant is used. In this algorithm, a batch B of random samples is used to compute $\mathbf{g} = \frac{1}{|B|}\sum_{(x,y)\in B}\nabla_\theta J(\theta; x)$, and estimation to the gradient $\nabla_\theta J(\theta)$.
In order to protect the privacy of the training data, it can be modified the SGD algorithm: after computing $\nabla_\theta J(\theta, x)$ for a batch B, we clip the l_2 norm of each gradient, compute the average, add noise and update the weights in a direction slightly modified by the noise. This approach can be studied in detail in [1], where the computation of the overall privacy cost (ϵ, δ) is explained.

Algorithm 6 DP-SGD

Input: Data $\{(x_i, y_i)\}_{i=1}^N$, a loss function $J(\theta)$, a learning rate η_t, a noise scale σ and a gradient norm bound C

1: Initialize θ randomly.
2: **for** each step t **do**
3: Take a batch $B \subset \mathcal{B}$ of samples
4: For sample i in B, compute $g_t(x_i) = \nabla_{\theta_t} J(\theta_t, x_i)$
5: Clip gradient: $g_t(x_i) = g_t(x_i) / \max\big(1, \frac{||g_t(x_i)||_2}{C}\big)$
6: Add noise: $g_t = \frac{1}{|B|}\big(\sum_i g_t(x_i) + N(0, \sigma^2 C^2 \boldsymbol{I})\big)$
7: $\theta_{t+1} = \theta_t - \eta_t g_t$

A direct implementation of Algorithm 6 would be impractical in frameworks like PyTorch or Tensorflow, since they don't expose intermediate computations, including per-sample gradients (only give access to the gradients averaged over a batch). Instead of bypass this limitation with batch sizes of 1 (which would be quite inefficient), we recommend the framework Opacus[2] explained in [52], which makes the implementation of this algorithm (and other DP related algorithms) trivial to implement[3].

[2] `https://github.com/pytorch/opacus`

[3] For example, the implementation of Algorithm 6 using a MLP on MNIST dataset can be seen in `https://github.com/pytorch/opacus/blob/main/examples/mnist.py`

4.1.4 Applications of Differential Privacy

Differential Privacy has a wide range of applications, particularly in scenarios where sensitive data is analyzed. Some notable examples include:

- **Statistical Databases:** Differential Privacy can be used to release summary statistics from sensitive databases without compromising individual privacy.
- **Machine Learning:** Differential Privacy techniques can be applied to train machine learning models on sensitive data, ensuring that the models do not reveal information about individual training examples.
- **Census Data:** The US Census Bureau has adopted Differential Privacy to protect the privacy of respondents while releasing demographic statistics [5].

By incorporating Differential Privacy, organizations can provide robust privacy guarantees while still gaining valuable insights from their data. The balance between privacy and utility is crucial, and Differential Privacy offers a principled approach to achieving this balance.

4.1.5 Challenges and Considerations

While Differential Privacy offers strong privacy guarantees, it also presents several challenges:

- **Parameter Selection:** Choosing appropriate values for ϵ and δ is critical and often context-dependent. Lower values provide better privacy but can significantly impact data utility.
- **Sensitivity Calculation:** Accurately determining the sensitivity of a query is essential for effective implementation. Overestimating sensitivity can lead to excessive noise, while underestimating it can compromise privacy.
- **Complex Queries:** Implementing Differential Privacy for complex queries, such as those involving joins or non-linear transformations, can be challenging. The Fundamental Law of Information Recovery states that overly accurate answers to too many questions will destroy privacy (see Section 8.1 of [11] for a formal proof)

Despite these challenges, Differential Privacy remains a powerful tool for privacy-preserving data analysis. Ongoing research and development continue to enhance its practicality and effectiveness, making it a cornerstone of modern privacy-preserving technologies.

4.2 Generative Models

In Privacy Enhancing Machine Learning, generative models are being used to capture joint distributions of data in order to obtain synthetic samples without using real data. This can be used effectively to initialize deep learning models, perform data analysis, or apply it in competitions where models are initially validated using synthetic data and later tested (privately) with the original data.
The approach of this section will be to test the feasibility of using generative adversarial networks to extract data features and approximate their distribution. This is not done to train discriminative models with synthetic data, but rather to use synthetic data for tuning and designing models, thus enabling remote training (the model is trained locally where the data reside) without needing access to the original data, following the philosophy of Federated Learning (see section 2). Comparing and evaluating the design of a Federated Learning model is beyond the scope of this work; instead, the distributions of the original datasets and synthetic datasets will be compared and evaluated.
In summary, two datasets will be used to test the feasibility of using generated data to facilitate model training in a Federated Learning environment, without necessarily using this data in the final model training. The

aim is to approximate the distribution of the original dataset (a very challenging task). This could be done in conjunction with other techniques such as data anonymization, but this work will focus solely on generative models, particularly GANs. To assess the quality of synthetic data, scatter plots, histograms, and metrics measuring similarity in marginal distributions will be used.

The GAN (Generative Adversarial Network) architecture was originally proposed in [18]. Since then, there have been numerous variations and improvements, primarily focused on image generation. Although other generative models exist, such as diffusion models, the GAN architecture remains employed and studied. In its original definition, the architecture consists of two models: a generative model G that approximates the data distribution, and a discriminator model D that estimates the probability that a sample is real or synthetic (generated by G). Both G and D are MLP (Multi-Layer Perceptron) architectures; in the case of G, the output dimension must match that of the real data. The main idea is that during training, these two models compete: G seeks to learn to generate realistic data that can deceive D, while D aims to correctly distinguish between real and generated samples from G. The input vector for G consists of random noise, for example, $z \sim N(\mathbf{0}, \sigma\mathbf{I})$; denoted as $G = G(z) = G(z; \theta_g)$, where θ_g represents the network weights. The discriminator takes as input a real or synthetic data vector x, denoted interchangeably as $D = D(x) = D(x; \theta_d)$. The (simplified) training process is as follows:

First, the weights of G and D are initialized randomly. During each epoch, real data $x_1, ..., x_m$ and random noise $z_1, ..., z_m$ are sampled, where x_i and z_i have the same dimensions, for example, $x_i, z_i \in \mathbb{R}^p, \quad p \in \mathbb{N}$. In the case of tabular data, it can occur, for instance, $x_i, z_i \in \mathbb{R}^p \times \mathbb{Z}^k, \quad p, k \in \mathbb{N}$, if the table consists of p columns of real data and k columns of integer data. It can also occur that the data are categorical: if the i-th column is categorical, the i-th component of the vector x and z will take values in a finite set Ω that can be numerically encoded. This will affect the exact definition of the initial (a priori) distribution of z.

Next, the batch of synthetic samples $G(z_1), ..., G(z_m)$ is computed. Now D receives both the batch of real data $x_1, ..., x_m$ and the batch of synthetic data $G(z_1), ..., G(z_m)$ and must distinguish between them. Therefore, D is a discriminative (supervised) binary classification model; it classifies as 1 the data it confidently predicts as real and as 0 the synthetic data (D outputs a probability between 0 and 1). Based on this classification, the discriminator's loss function is calculated, and D's weights are adjusted to **minimize** this loss. After this step, another batch of synthetic data $G'(z_1), ..., G'(z_m)$ is generated, and G's loss is computed as it tries to deceive the discriminator (which has already been updated). G's weights are updated to **maximize** this loss. This process is repeated iteratively until a stopping criterion is met, such as a maximum number of epochs.

Therefore, this involves the following min-max game:

$$\min_G \max_D V(G, D) = E_{x \sim \rho_{\text{data}}(x)}[\log D(x)] + E_{z \sim \rho_z(z)}[\log(1 - D(G(z)))] \tag{4.1}$$

The intuitive idea behind this formula is as follows: the ultimate goal of the architecture is for ρ_z to converge to ρ_{data}, meaning that G should approximate the theoretical data distribution perfectly (although ρ_{data} is unknown and we only have a finite sample of data). To achieve this, we use D: if there were a perfect discriminator that could distinguish between real and synthetic data, we could use it to update the weights of G and improve the generative model. The approach is to train both models simultaneously with the aim of achieving perfect models: where D correctly classifies the data, maximizing the probability of classifying samples drawn from ρ_{data} as real, and minimizing the probability of classifying samples generated by G as real. This is expressed as:

$$\max_D V(G, D) = E_{x \sim \rho_{\text{data}}(x)}[\log D(x)] + E_{z \sim \rho_z(z)}[\log(1 - D(G(z)))]$$

At the same time, given a discriminator D, we want G to maximize the probability that D classifies samples drawn from ρ_z as real. This can be formulated as:

$$\max_G V(G, D) = E_{z \sim \rho_z}[\log D(G(z))] \Leftrightarrow \min_G V(G, D) = E_{z \sim \rho_z}[\log(1 - D(G(z)))]$$

These two formulations lead us to (4.1).

Note: In implementations, care is taken to ensure that the probability returned by D is not exactly 0 or 1

to avoid log from approaching 0.

Taking this into account, we can now understand the expressions given in the original paper [18] for updating the model weights. Let's start with D, which receives batches of real data $x_1, ..., x_m$ and synthetic data $G(z_1), ..., G(z_m)$. Stochastic gradient ascent is applied as follows:

$$\theta'_d = \theta_d + \eta \frac{\partial}{\partial \theta_d} \left[\frac{1}{m} \sum_{i=1}^{m} (\log D(x_i) + \log(1 - D(G(z_i)))) \right]$$

Note: We take the mean of individual losses, which represents the empirical risk: an estimation of the theoretical risk since we do not know the theoretical data distribution. I emphasize the + sign, as our goal is to maximize.

For G, the weights are updated using stochastic gradient descent as follows:

$$\theta'_g = \theta_g - \eta \frac{\partial}{\partial \theta_g} \left[\frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i))) \right]$$

Note: This is a simplified version of stochastic gradient methods. The original paper mentions using a variant with momentum (see section 1.4). There are many variations with different hyperparameters and features. A deliberately simplified terminology has been maintained.

Since both G and D are MLPs, gradients can be efficiently computed using the backpropagation algorithm.

For tabular datasets, certain particularities must be addressed ([48]). In particular, we will use a variant of the original GAN: the Conditional GAN. We will consider features (columns) of both continuous and discrete data. The generator model must differentiate between the two; in the final layer, a $softmax$ activation function will be used for the discrete case and $tanh$ for the continuous case.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad x \in \mathbb{R}$$
$$softmax(\mathbf{x})_j = \frac{e^{x_j}}{\sum i = 1^n e^{x_i}} \quad j = 1, ..., n \quad \mathbf{x} \in \mathbb{R}^n$$

In a conditional GAN, both the generator and the discriminator also receive additional information, which can be in the form of class labels, specific features, or any other relevant information for the problem at hand. This additional information is provided as a condition for the generation of synthetic samples and the evaluation of their authenticity.

Introducing conditional information into a GAN allows for more specific control over the generation process. For example, in the case of image generation conditioned on certain class labels, a conditional GAN can generate images of a specific type, such as "cats" or "dogs," according to the label provided as the condition. In the context of CTGAN (Conditional Tabular Generative Adversarial Networks), this additional information can be any relevant feature or set of features present in the original data. For example, if working with a tabular dataset containing information about bank customers, the additional information could include the customer's gender, age, account type, credit history, etc.

During the training process of the CTGAN, both the generator and the discriminator receive this additional information as a condition for the generation and evaluation of synthetic data. This means that the generator learns to generate synthetic samples that are not only statistically similar to the original data but also conditioned on specific values of the provided features.

For example, if one wants to generate synthetic data of bank customers with a certain age range and gender, this information can be provided as a condition to the CTGAN during the generation process. As a result, the CTGAN will be able to generate synthetic samples that follow the distributions of the conditioned features, which can be useful in various applications, such as generating balanced training data, preserving privacy, or conducting sensitive testing. In summary, CTGAN applies the principles of conditional GANs to

the specific context of tabular data synthesis.
When working with tabular data instead of images, several aspects must be considered, such as continuous variables not typically following a normal distribution; in fact, they may follow a multimodal distribution (having more than one peak of maximum density). Additionally, there can be issues of imbalanced features (for example, in the case of banking data, a binary variable determining default status is often very imbalanced, as most customers are not defaulters).
To numerically evaluate the synthetic datasets, the KSComplement and TVComplement metrics from the *SDMetrics* library will be used. The KSComplement metric calculates the similarity (of the marginal distribution) between two columns, one real and one synthetic. It uses the Kolmogorov-Smirnov statistic, which measures the maximum difference between two cumulative probability distributions, yielding a value between 0 and 1. KSComplement returns $1 - KS$, meaning the closer to 1, the more similar the distributions. This metric will be used for continuous variables. For categorical variables, the TVComplement metric from the same library will be used, which also measures the similarity between two marginal distributions. It calculates the frequentist probability of each class and sums the absolute differences for each class. The metric returns $1 -$ result, so the closer the value is to 1, the better the distributions match.

Results:

For the implementation, the CTGAN library in Python has been used. This library addresses the problem of non-Gaussian and multimodal data through variational Gaussian mixture models to estimate complex distributions. Explaining Gaussian mixtures is beyond the scope of this work, but [4] has been used as a reference for this type of model.
A total of 2 datasets have been used to check the performance of CTGAN. The first consists of a dataset generated by myself, composed of two columns x and y that follow a normal distribution $N(\mu, \Sigma)$, where $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$, meaning they are positively correlated variables. The third column is a categorical feature with 5 classes encoded with numbers from 0 to 4. These classes have been generated using the probability vector $p = (0.3, 0.2, 0.4, 0.07, 0.03)$. A thousand samples of this dataset have been generated, and CTGAN was trained for 3000 epochs. This is a somewhat arbitrary stopping criterion after observing the objective function graphs, as the loss curves of G and D oscillate and do not reach Nash equilibrium. Theoretically, the optimum is reached when $D = 1/2$, that is, when it is unable to differentiate between real and synthetic data and determines this randomly. The goal of this work is not to train the best possible model but to study its feasibility according to the privacy objectives set.

A thousand rows of ρ_z were sampled:

	x	y	Var_Discreta
0	-1.062465	-0.214028	2
1	-0.644456	-0.019780	1
2	-0.714935	-1.175347	0
3	0.188888	-0.037017	0
4	1.113975	-0.326813	0

Figure 4.2: Real samples

	x	y	Var_Discreta
0	1.493901	-1.629266	0
1	-0.549035	-4.325017	3
2	0.073771	-0.715224	4
3	2.152546	2.460955	4
4	0.799019	1.267281	0

Figure 4.3: Synthetic samples

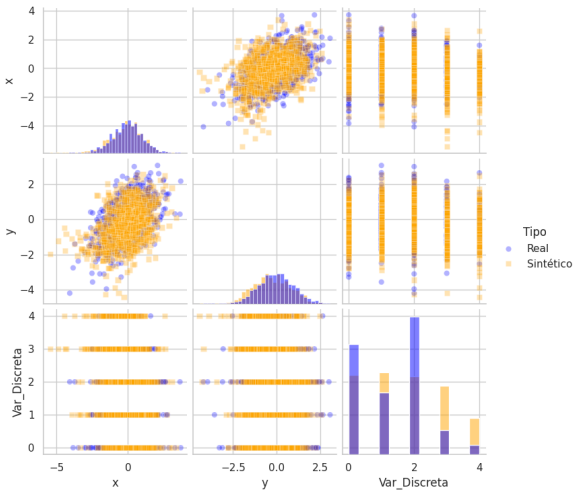

Figure 4.4: Scatter plot and histogram.

- KS-Complement x: 0.94
- KS-Complement y: 0.887
- TV-Complement discrete variable: 0.773

For the second test, the Iris dataset has been used, which contains 4 continuous features and 1 categorical feature (encoded as a discrete variable from a finite set). This dataset contains few samples (150 in total), so the estimation of the distribution may be affected by the small amount of real data. It was trained for 1500 epochs in a few seconds. The 'species' variable was encoded as { 'setosa': 0, 'versicolor': 1, 'virginica': 2 }. After training, 150 data points were sampled from the generated distribution:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Figure 4.5: Real samples

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.547129	3.268097	0.898987	0.179233	setosa
1	5.016506	2.274413	3.366716	0.867121	versicolor
2	7.019291	3.168845	5.481823	1.666999	virginica
3	4.855492	3.073170	1.235019	0.084497	setosa
4	5.981183	3.126868	4.614439	2.017656	virginica

Figure 4.6: Synthetic samples

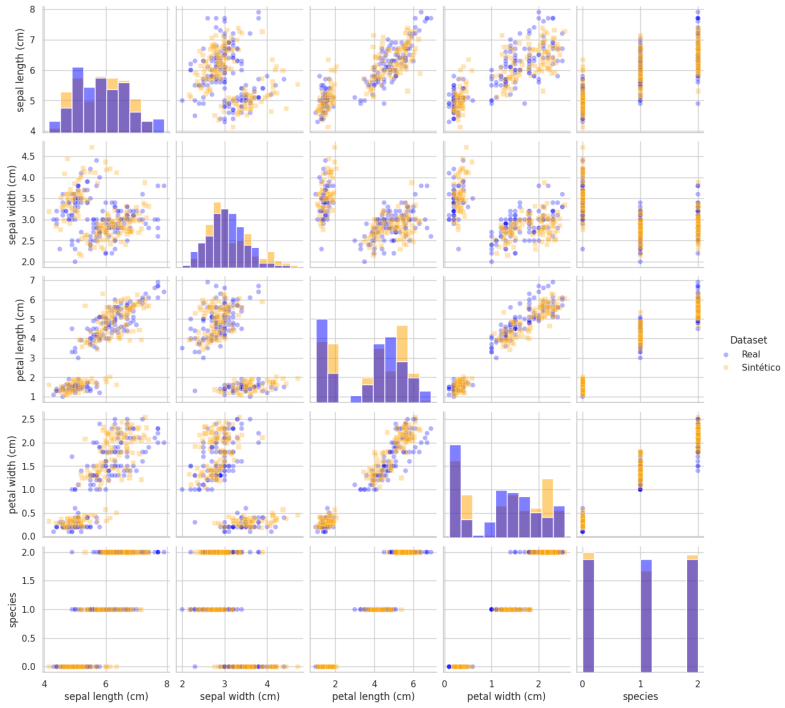

Figure 4.7: Scatter plot and histogram.

- KS-Complement sepal length (cm): 0.91
- KS-Complement sepal width (cm): 0.85
- KS-Complement petal length (cm): 0.89
- KS-Complement petal width (cm): 0.8
- TV-Complement species: 0.96

We observe that in both datasets, a high marginal similarity of the columns has been achieved (close to 1). The scatter plots show that the model has relatively well captured the correlations between variables, and the histograms between real and synthetic data are quite similar. However, good performance in these datasets does not necessarily imply it is a good model. It is currently known that there are models that better capture the distribution of tabular data, although the use of GANs can still be useful in this task. Available resources, such as computing power or the amount of data, will guide the choice between different methods.

The generation of synthetic tabular data using GANs can be an additional tool in the set of technologies aiming to train models with sensitive data while safeguarding privacy. The quality of synthetic data depends on many factors: the original dataset size, imbalanced classes, etc. However, these networks seem to approximate the distribution of the original data, though not perfectly. Approximating the joint distribution is a very complex problem to solve, and in this work, only very simple datasets have been used, and even with these datasets, it has not been possible to perfectly approximate the original distribution.

This is why, although it is a promising technique, I do not believe it is appropriate to use it for conducting scientific studies or training models solely using synthetic data. Nonetheless, they do fulfill the intended function: serving as a guide for adjusting models in a Federated Learning environment without needing to share the original data. With synthetic data, the different entities that hold the data can share these synthetic data to help researchers understand, albeit approximately, the type of data they need to work with.

For this purpose, I believe that data generation can be helpful. It should be studied to what extent generative models can resist attacks aimed at extracting information about the original data ([42]), although this is beyond the scope of this work.

Other techniques for generating synthetic data use architectures different from GANs, such as Variational Autoencoders [48] or simpler statistical models like Gaussian Copulas [26]. For a detailed summary of the various tabular data generation techniques, I recommend checking [31].

Conclusion

Different methods of preserving data privacy during the training of Machine Learning models have been studied. The aim has been to have a general overview of what is being used in practice and what challenges and drawbacks each method has.
Federated Learning makes sense in particular cases where different entities want to collaborate to improve models using data they could not have obtained otherwise. However, if the training and data distribution are not supervised, the resulting model may not perform as expected. In other cases, it is not necessary to train neural network models, and simpler models like decision trees or XGBoosting may be more appropriate. For these cases, there are cryptographic protocols within the field of Multi-Party Computation that allow for secure and private training without significant performance loss, albeit at the cost of increased computational and communication overhead.
It has been seen that implementing a SplitNN model can make sense in cases where the data has a vertical partition, thanks to protocols such as Private Set Intersection. Other VFL models require the use of cryptographic protocols or MPC. Again, this solves particular cases, not just any problem where the data follows this type of partition.
Methods like Differential Privacy also have use cases, such as obtaining statistical metrics by querying databases or enhancing the security of deep learning models during the optimization step. However, this comes at a performance cost to the model, which must be adjusted according to the desired level of privacy.
Generative AI, despite its surge in recent years, does not immediately solve the privacy problem either, as it is known that original data can be obtained from synthetic samples. It can be useful in specific cases, understanding the privacy guarantees sought.

Data generates money and value, and thus there will always be an interest in exploiting it. The fact that public institutions are regulating and punishing practices that circumvent indiscriminate use of data is an indicator that there is a growing concern about the use of personal data to feed machine learning models, often privately in companies and institutions that are not transparent about their practices. Industries and universities must continue exploring different techniques to preserve the privacy of citizens. Still, we must contextualize each technique with its use case and understand the tools we are working with.

In addition, we must not only be concerned with the protection of the data with which they are trained, but we must also study the possible discriminatory biases they may have, the robustness, and the explainability of these models. All these challenges will not be solved with a revolutionary algorithm that fixes data privacy, model security, or biases. These are complex issues that need to be studied from diverse perspectives in collaborative environments, as they involve not only technical problems but also social and legislative ones. We must not impose a technical solution on a poorly conceived problem.
The technologies presented here are promising but not magical. They require research, careful implementation, and supervision. They are not free from criticism and possible security flaws in certain cases, in addition to a potential degradation of models compared to centralized training. The aim was not to present these topics as the solution to privacy, but rather to understand, even if superficially, what they are about and what they seek to solve. I hope I have achieved this, or at least provided a good bibliography to consult material on each topic.

Appendix A

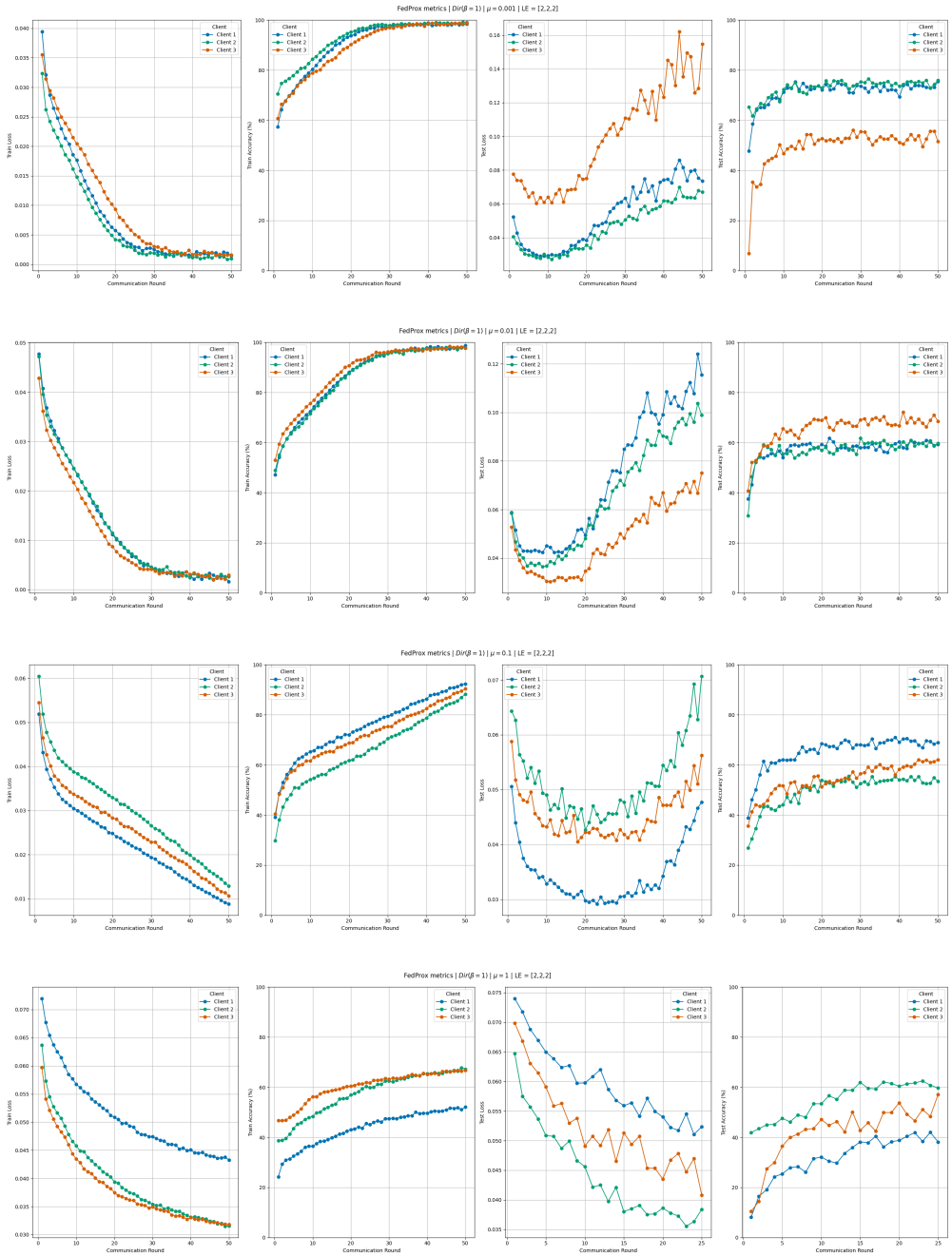

Figure 4.8: Local metrics for 3 clients in 50 communication rounds using FedProx with a Non-IID setting over the CIFAR10 dataset, $\mu \in \{0.001, 0.01, 0.1, 1\}$. Label distribution skew using the Dirichlet distribution with $\boldsymbol{\beta} = (1, 1, 1)$

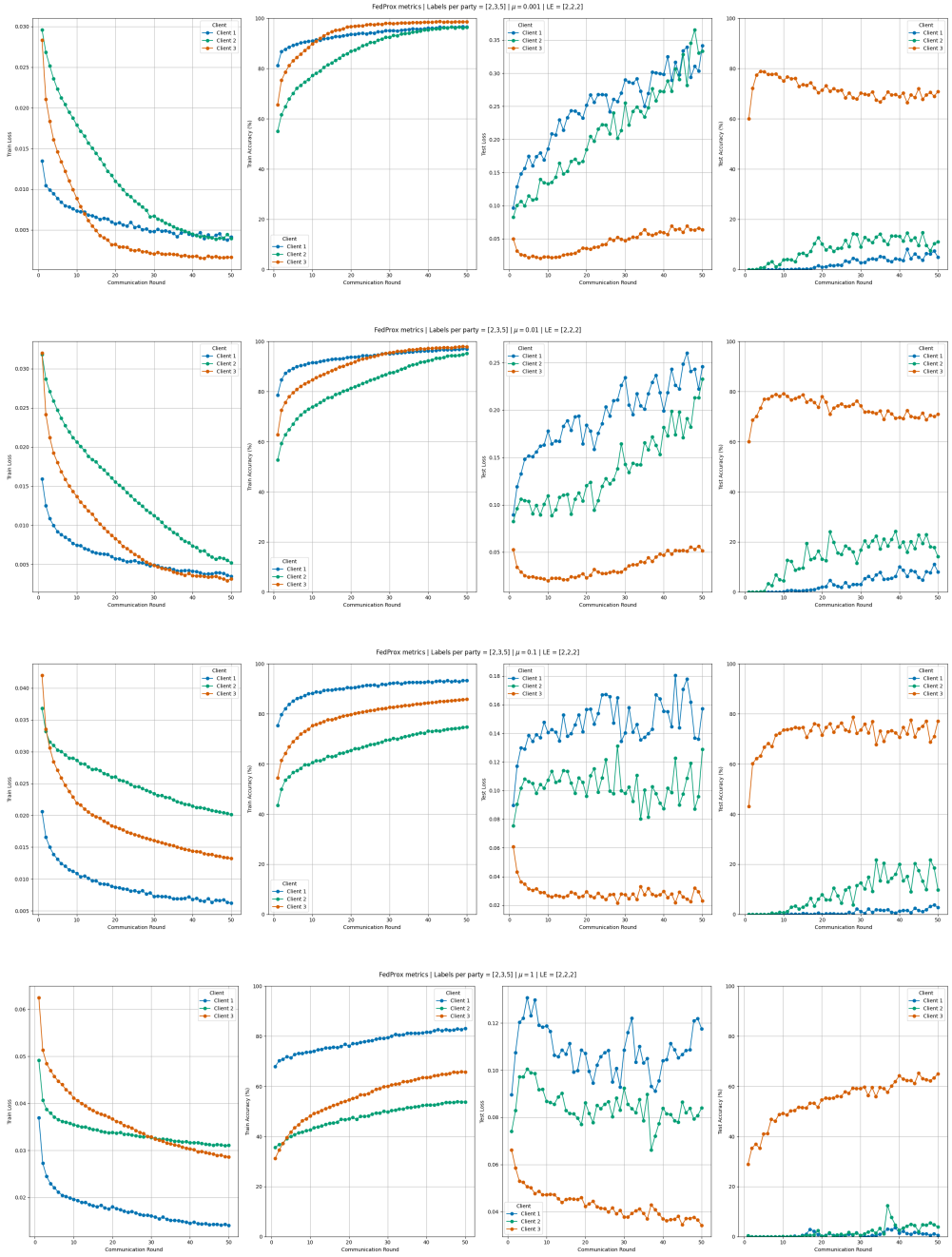

Figure 4.9: Local metrics for 3 clients in 50 communication rounds using FedProx with a Non-IID setting over the CIFAR10 dataset, $\mu \in \{0.001, 0.01, 0.1, 1\}$. Label distribution skew, the first client data from 2 classes, the second client from 3 classes and the third client from the 5 remaining classes.

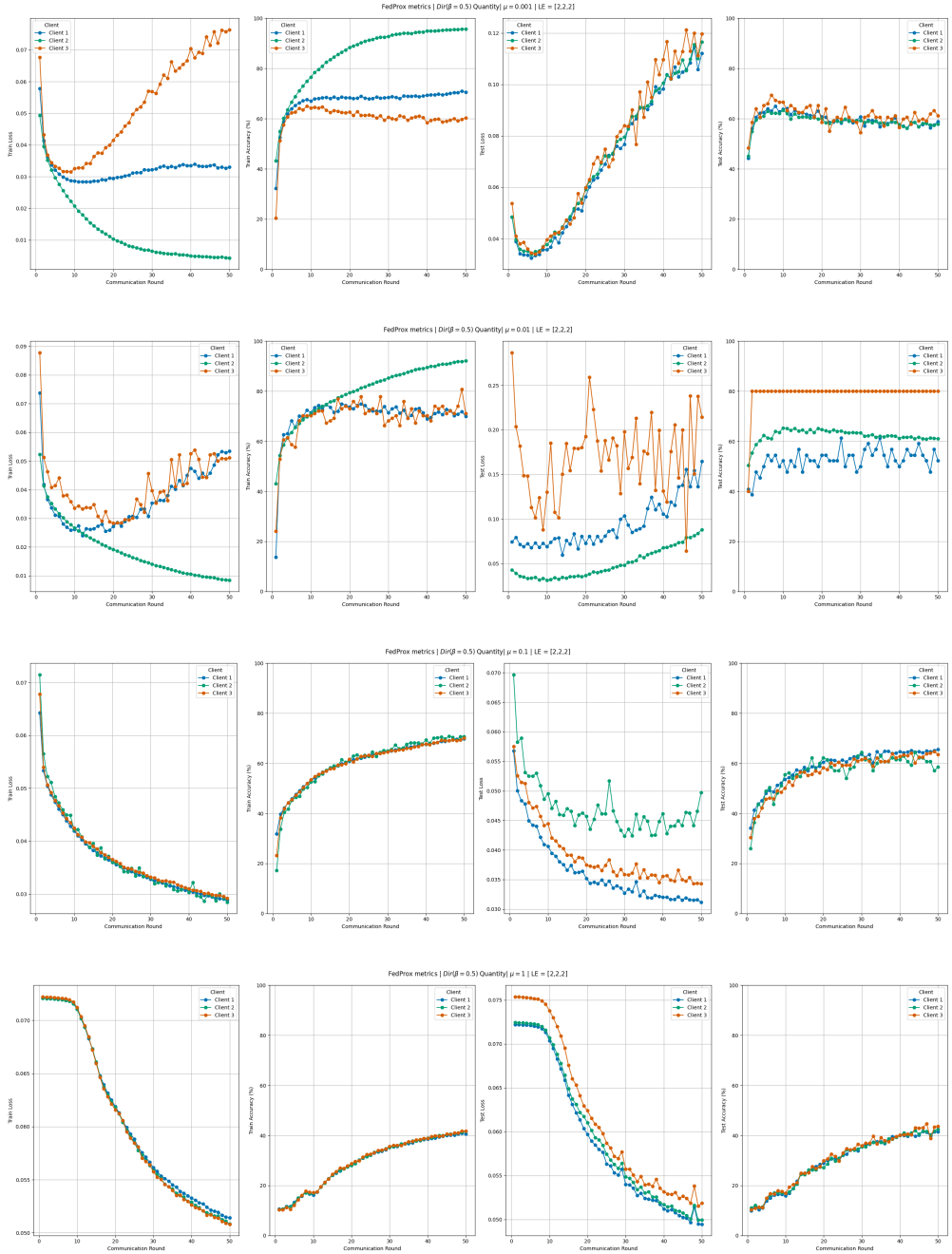

Figure 4.10: Local metrics for 3 clients in 50 communication rounds using FedProx with a Non-IID setting over the CIFAR10 dataset, $\mu \in \{0.001, 0.01, 0.1, 1\}$. Label quantity distribution skew using $Dir(\boldsymbol{\beta})$, $\boldsymbol{\beta} = (0.5, 0.5, 0.5)$

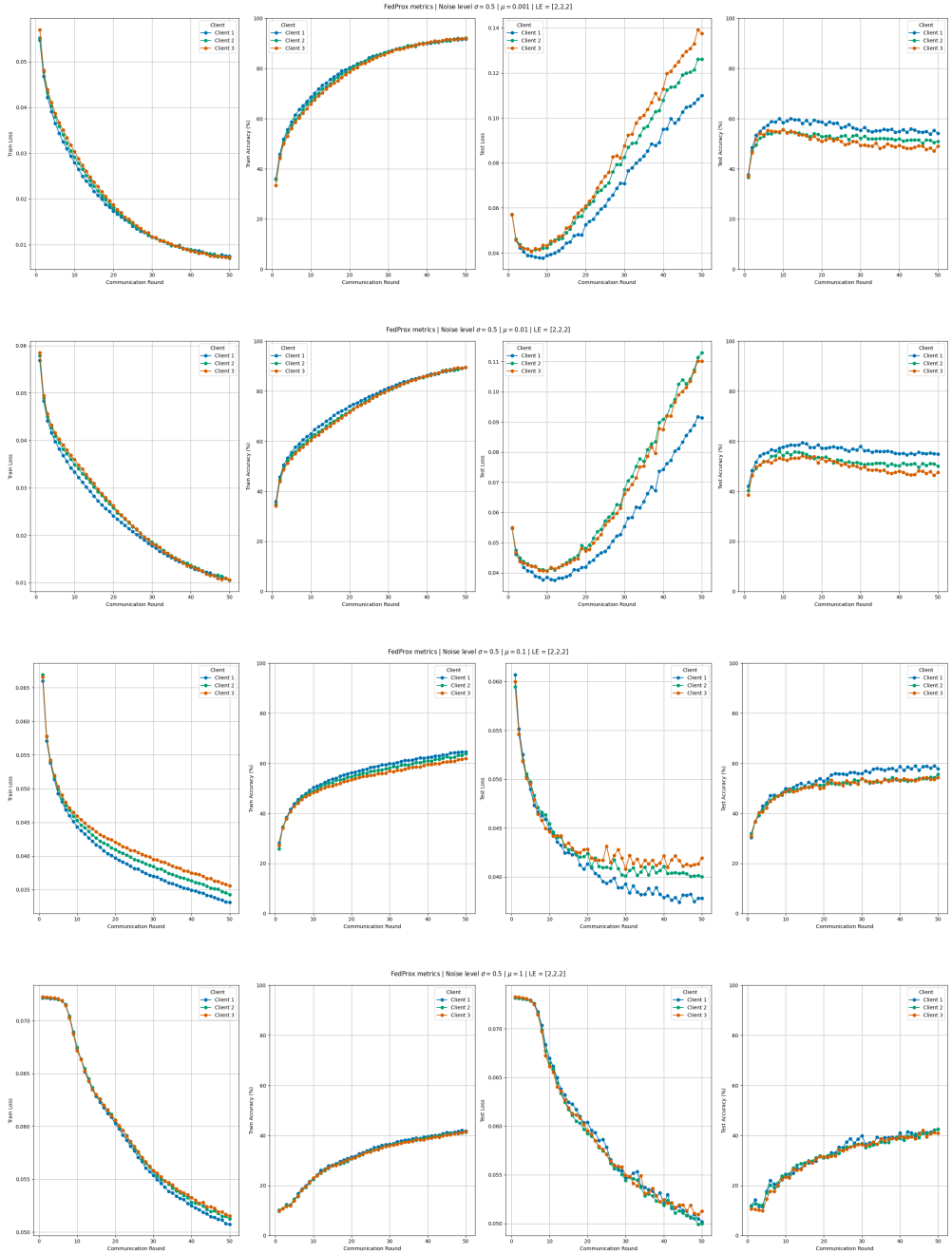

Figure 4.11: Local metrics for 3 clients in 50 communication rounds using FedProx with a Non-IID setting over the CIFAR10 dataset, $\mu \in \{0.001, 0.01, 0.1, 1\}$. Feature distribution skew using with noise level $\sigma = 0.5$

Bibliography

[1] Martín Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Oct. 2016, pp. 308–318. DOI: 10.1145/2976749.2978318. arXiv: 1607.00133 [cs, stat]. (Visited on 07/21/2024).

[2] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. "Information sharing across private databases". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, pp. 86–97.

[3] *Así Es La Nueva Ordenanza Tipo de La FEMP Sobre Gobierno Del Dato | Datos.Gob.Es*. URL: https://datos.gob.es/es/blog/asi-es-la-nueva-ordenanza-tipo-de-la-femp-sobre-gobierno-del-dato (visited on 07/27/2024).

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer, 2006. ISBN: 978-0-387-31073-2.

[5] US Census Bureau. *Why the Census Bureau Chose Differential Privacy*. URL: https://www.census.gov/library/publications/2023/decennial/c2020br-03.html (visited on 07/18/2024).

[6] Ovidiu Calin. *Deep Learning Architectures: A Mathematical Approach*. Springer Series in the Data Sciences. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-36720-6 978-3-030-36721-3. DOI: 10.1007/978-3-030-36721-3. (Visited on 05/23/2024).

[7] Saeed Damadi, Golnaz Moharrer, and Mostafa Cham. *The Backpropagation Algorithm for a Math Student*. May 2023. arXiv: 2301.09977 [cs, math]. (Visited on 05/26/2024).

[8] *Data Act | Shaping Europe's Digital Future*. URL: https://digital-strategy.ec.europa.eu/en/policies/data-act (visited on 07/27/2024).

[9] Yann Dauphin et al. *Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization*. June 2014. arXiv: 1406.2572 [cs, math, stat]. (Visited on 05/24/2024).

[10] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 141–142. ISSN: 1558-0792. DOI: 10.1109/MSP.2012.2211477. (Visited on 07/18/2024).

[11] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3-4 (2013), pp. 211–407. ISSN: 1551-305X, 1551-3068. DOI: 10.1561/0400000042. (Visited on 07/18/2024).

[12] Daniel Escudero. *An Introduction to Secret-Sharing-Based Secure Multiparty Computation*. 2022. (Visited on 07/01/2024).

[13] David Evans, Vladimir Kolesnikov, and Mike Rosulek. "A Pragmatic Introduction to Secure Multi-Party Computation". In: *Foundations and Trends® in Privacy and Security* 2.2-3 (2018), pp. 70–246. ISSN: 2474-1558. DOI: 10.1561/3300000019. URL: http://dx.doi.org/10.1561/3300000019.

[14] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Denver Colorado USA: ACM, Oct. 2015, pp. 1322–1333. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813677. (Visited on 07/20/2024).

[15] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, June 2011, pp. 315–323. (Visited on 05/23/2024).

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0-262-03561-3.

[18] Ian J. Goodfellow et al. *Generative Adversarial Networks*. June 2014. arXiv: 1406.2661 [cs, stat]. (Visited on 07/14/2024).

[19] Otkrist Gupta and Ramesh Raskar. *Distributed Learning of Deep Neural Network over Multiple Agents*. Oct. 2018. arXiv: 1810.06060 [cs, stat]. (Visited on 07/17/2024).

[20] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. "PATE-GAN: Generating synthetic data with differential privacy guarantees". In: *International conference on learning representations*. 2018.

[21] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. Mar. 2021. arXiv: 1912.04977 [cs, stat]. (Visited on 05/15/2024).

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. arXiv: 1412.6980 [cs]. (Visited on 05/26/2024).

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386. (Visited on 05/23/2024).

[24] Qinbin Li et al. *Federated Learning on Non-IID Data Silos: An Experimental Study*. Oct. 2021. arXiv: 2102.02079 [cs]. (Visited on 01/12/2024).

[25] Tian Li et al. *Federated Optimization in Heterogeneous Networks*. Apr. 2020. arXiv: 1812.06127 [cs, stat]. (Visited on 12/16/2023).

[26] Zheng Li, Yue Zhao, and Jialin Fu. *SYNC: A Copula Based Framework for Generating Synthetic Data from Aggregated Sources*. Sept. 2020. arXiv: 2009.09471 [cs, stat]. (Visited on 08/17/2024).

[27] Yunhui Long et al. "G-PATE: Scalable Differentially Private Data Generator via Private Aggregation of Teacher Discriminators". In: ().

[28] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. Jan. 2023. arXiv: 1602.05629 [cs]. (Visited on 01/12/2024).

[29] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning. Cambridge, Massachusetts London, England: The MIT Press, 2022. ISBN: 978-0-262-36930-5 978-0-262-04682-4.

[30] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. Dec. 2015. arXiv: 1511.08458 [cs]. (Visited on 05/26/2024).

[31] Eugenia Papadaki, Aristidis G. Vrahatis, and Sotiris Kotsiantis. "Exploring Innovative Approaches to Synthetic Tabular Data Generation". In: *Electronics* 13.10 (Jan. 2024), p. 1965. ISSN: 2079-9292. DOI: 10.3390/electronics13101965. (Visited on 08/17/2024).

[32] Nicolas Papernot et al. *Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data*. Mar. 2017. arXiv: 1610.05755 [cs, stat]. (Visited on 07/18/2024).

[33] Ning Qian. "On the Momentum Term in Gradient Descent Learning Algorithms". In: *Neural Networks: The Official Journal of the International Neural Network Society* 12.1 (Jan. 1999), pp. 145–151. ISSN: 1879-2782. DOI: 10.1016/s0893-6080(98)00116-6.

[34] Sashank Reddi et al. *Adaptive Federated Optimization*. Sept. 2021. arXiv: 2003.00295 [cs, math, stat]. (Visited on 07/23/2024).

[35] *Regulation - EU - 2024/1689 - EN - EUR-Lex*. https://eur-lex.europa.eu/eli/reg/2024/1689/oj. (Visited on 08/30/2024).

[36] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. (Visited on 05/22/2024).

[37] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. June 2017. arXiv: 1609.04747 [cs]. (Visited on 05/24/2024).

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. (Visited on 05/26/2024).

[39] Aradhana Sahu and Samarendra Ghosh. *Review Paper on Secure Hash Algorithm With Its Variants*. Tech. rep. May 2017. DOI: 10.13140/RG.2.2.13855.05289.

[40] Reza Shokri et al. *Membership Inference Attacks against Machine Learning Models*. Mar. 2017. arXiv: 1610.05820 [cs, stat]. (Visited on 07/20/2024).

[41] Carl Smestad and Jingyue Li. "A Systematic Literature Review on Client Selection in Federated Learning". In: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. June 2023, pp. 2–11. DOI: 10.1145/3593434.3593438. arXiv: 2306.04862 [cs]. (Visited on 05/11/2024).

[42] Hui Sun et al. "Adversarial Attacks Against Deep Generative Models on Data: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 35.4 (Apr. 2023), pp. 3367–3388. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2021.3130903. (Visited on 07/14/2024).

[43] Ruoyu Sun. *Optimization for Deep Learning: Theory and Algorithms*. Dec. 2019. arXiv: 1912.08957 [cs, math, stat]. (Visited on 05/24/2024).

[44] Shiliang Sun et al. *A Survey of Optimization Methods from a Machine Learning Perspective*. Oct. 2019. arXiv: 1906.06821 [cs, math, stat]. (Visited on 05/24/2024).

[45] Jianyu Wang et al. *A Field Guide to Federated Optimization*. July 2021. arXiv: 2107.06917 [cs]. (Visited on 05/14/2024).

[46] Jianyu Wang et al. *Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization*. July 2020. arXiv: 2007.07481 [cs, stat]. (Visited on 05/15/2024).

[47] Alexandra Wood et al. "Differential Privacy: A Primer for a Non-Technical Audience". In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3338027. (Visited on 07/20/2024).

[48] Lei Xu et al. "Modeling Tabular Data Using Conditional GAN". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 659. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 7335–7345. (Visited on 07/13/2024).

[49] Nesterov Y. "A Method of Solving a Convex Programming Problem with Convergence Rate O(1/K**2)". In: *Doklady Akademii Nauk SSSR* 269.3 (1983), p. 543. (Visited on 05/25/2024).

[50] Liu Yang et al. *A Survey on Vertical Federated Learning: From a Layered Perspective*. Apr. 2023. arXiv: 2304.01829 [cs]. (Visited on 08/17/2024).

[51] Zhengjie Yang et al. "Federated Learning with Nesterov Accelerated Gradient". In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (Dec. 2022), pp. 4863–4873. ISSN: 1045-9219, 1558-2183, 2161-9883. DOI: 10.1109/TPDS.2022.3206480. arXiv: 2009.08716 [cs, stat]. (Visited on 07/23/2024).

[52] Ashkan Yousefpour et al. *Opacus: User-Friendly Differential Privacy Library in PyTorch*. Aug. 2022. arXiv: 2109.12298 [cs]. (Visited on 07/21/2024).

[53] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. Dec. 2012. arXiv: 1212.5701 [cs]. (Visited on 05/26/2024).

[54] Yuanbo Zhang et al. *Private Federated Learning in Gboard*. June 2023. arXiv: 2306.14793 [cs]. (Visited on 05/11/2024).

[55] Zhilu Zhang and Mert R. Sabuncu. *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels*. Nov. 2018. arXiv: 1805.07836 [cs, stat]. (Visited on 05/24/2024).

[56] Yue Zhao et al. "Federated Learning with Non-IID Data". In: (2018). DOI: 10.48550/arXiv.1806.00582. arXiv: 1806.00582 [cs, stat]. (Visited on 01/12/2024).

[57] Hangyu Zhu et al. *Federated Learning on Non-IID Data: A Survey*. June 2021. arXiv: 2106.06843 [cs]. (Visited on 08/17/2024).