

Faculté des sciences

Ecole de Statistique, Biostatistique et Sciences Actuarielles

Federated Learning

Introduction to privacy-preserving
collaborative machine learning

Author : Lionel LAMY

Supervisor : Robin VAN OIRBEEK

Reader : Christian HAFNER

Academic year : 2022 - 2023

Master [120] in Data Science : statistics

Table of Contents

List of Symbols	iii
List of Figures	iv
List of Tables	vi
I. Introduction	1
1 Federated learning	1
2 Research	3
3 Organization	4
II. Overview of Federated Learning	5
1 Definitions	5
2 Components	7
2.1 Nodes	7
2.2 Manager	9
3 Process	10
4 Challenges	12
4.1 Statistical heterogeneity	12
4.2 System heterogeneity	13
4.3 Evaluation and monitoring	14
4.4 Threats	14
4.5 Protections	16
III. Methods and Algorithms	17
1 Problem formulation	17
2 Solving the optimization problem	18
2.1 FedOpt : the framework	18
2.2 FedAvg : the baseline	20
3 Tackling the heterogeneity	23
3.1 FedProx : proximal term	23
3.2 FedNova : objective consistency	25
4 Incorporate server-side adaptivity	29
5 Adding the privacy layer	31
5.1 Differential privacy	31

IV.	Implementation & Research	34
1	Datasets	34
1.1	CIFAR-10	34
1.2	FEMNIST	36
1.3	Shakespeare	37
2	Experiments	38
2.1	Setup	38
2.2	Statistical heterogeneity analysis	42
2.3	Robustness to system heterogeneity	47
2.4	Extra: fine-tuning and utility of federated learning	55
V.	Conclusion	57
A.	Theoretical background	59
1	Assumptions on optimization	59
2	Gradient descent	60
3	Momentum optimizers	60
B.	Details	64
1	Algorithms	64
1.1	FedProx*	64
1.2	FedAdadelta	64
2	Tables	65
2.1	Resnet-18 architecture	65
2.2	TwoCNN architecture	66
2.3	NextCharLSTM architecture	67
2.4	CIFAR10 Resnet - Fairness - Data Heterogeneity	67
3	Figures	68
3.1	Flower framework	68
3.2	CIFAR10 - Individual accuracies on validation set	69
3.3	CIFAR10 - Resnet - Convergence speed comparison	70
3.4	CIFAR10 - Barplot comparison of models	72
3.5	FEMNIST - System heterogeneity - Comparison	73
3.6	FEMNIST - Fairness histograms - S09D05	74
3.7	Shakespeare - System heterogeneity - Comparison	75
3.8	Shakespeare - Fairness histograms - Basic	76

List of Symbols

Basics	ζ	Set of all possible clients
	$\mathcal{C}$	Meta-distribution of clients
	c	A particular client
	C	Amount of clients available
	S^t	Subset of selected clients at round t
	Ξ	Population data
	$\mathcal{D}_c$	Distribution of data of a client
	D_c	The set of samples of a client
	$\mathcal{B}$	Batch (of data)
	B	Batch size
	n_c	The amount of samples of a client
	ξ	A sample
	$\mathcal{X}$	Space of features in a sample
	$\mathcal{Y}$	Space of labels in a sample
	T	Amount of rounds
	E	Num of requested epochs
Optimisation	$F(\mathbf{w})$	The global objective
	$F_c(\mathbf{w})$	The local objective of a client
	$f_c(\xi, \mathbf{w})$	The loss function of a client using parameters on sample
	$\mathbf{w}$	Parameters (weights) of a model
	Δ	Model update (difference between two weights)
	η	The learning rate: η_c and η_s for client and server
	τ_c	The amount of local work done by a client
	$\mathbf{g}$	Gradient using gradient descent
Privacy	$\tilde{\mathbf{g}}$	Gradient using any optimizer
	ε	Privacy budget
	δ	Privacy leakage, failure probability
	M	Clip norm
	z	Noise multiplier
	γ	Target quantile

List of the most important and commonly used symbols

List of Figures

II.2	Taxonomy of FL Systems, by Li et al. (2021) [28]	10
II.3	Simplified illustration of the training process (cross-device). Vectorized, then modified from <i>What is Federated Learning</i> [46]	11
III.3	Illustration of FedNova against FedAvg for 2 clients, $\tau_1 = 4, \tau_2 = 2$. .	29
IV.1	CIFAR10 random samples, labels on top	35
IV.2	Samples of one writer, labels on top	36
IV.5	Amount of not-dropped clients per round. 10 clients selected and 100 rounds, with 30% and 50% (in expectation) of client droppped. Also valid for the stragglers selection mechanism.	39
IV.6	Illustration of common learning rates decay strategies, starting at 0.9. The exponential function has a factor of 0.95.	41
IV.9	Test and validation accuracies of FedAvg for the CIFAR10 dataset with 4 different sampling strategies and heterogeneity levels. The crosses at the bottom helps visualise when the algorithms goes over the threshold. The colored dots are the accuracies on the full set of clients. ResNet model.	43
IV.12	Barplot comparing the accuracy on the full test set, 100 rounds. Errors are the absolute difference between the full test and (biased) distributed evaluation.	45
IV.13	Comparison of the distributed test accuracy for the ResNet and TwoCNN models on the CIFAR10 (0.5-0.5) dataset.	47
IV.15	Accuracy convergence of algorithms on FEMNIST. Basic scenario. . .	49
IV.16	Accuracy convergence of algorithms on FEMNIST. Scenario with 90% stragglers and 50% of devices dropping out.	49
IV.17	Barplot of the best accuracy in 100 rounds on the full test set. . . .	50
IV.18	The 5 test samples of the client with the lowest accuracy	51
IV.19	Fairness histogram comparing FedAvg to FedNova and FedNova+Adam in the S09D05 scenario of FEMNIST. 3597 clients. Mean accuracies are shown with vertical lines. The color of FedNova+Adam is different for the sake of visibility.	52
IV.20	Accuracy convergence of algorithms on Shakespeare. Basic scenario. .	53
IV.21	Barplot of the best accuracy on the test set for the Shakespeare dataset. 100 rounds.	54

IV.22	Violin plot of the test accuracies for every client, before and after fine-tuning on the CIFAR10 dataset.	55
B.1	Diagram of the Flower process with 2 users	68
B.3	Comparison of the convergence speed in terms of test accuracy, ResNet-18, CIFAR10 (inf-0.5) dataset. Crosses and dots at the bottom indicates when the algorithms goes above the target for the distributed and full test sets, respectively. Accuracy on the validation set of the clients is shown in transparency.	70
B.4	Comparison of the convergence speed in terms of test accuracy (inf-inf).	70
B.5	Comparison of the convergence speed in terms of test accuracy (0.5-inf).	71
B.6	Comparison of the convergence speed in terms of test accuracy (0.5-0.5).	71
B.7	Comparison of the best full test set accuracies on CIFAR10 for both models. 100 rounds.	72

List of Tables

II.1	Adapted from Table 1 of Li et al. (2020), compares cross-device and cross-silo systems with the traditional datacenter distributed learning across various dimensions. These characteristics play a critical role in determining which algorithms and techniques are suitable for a given scenario.	8
IV.1	Satisfactory accuracies considered in our experiments and amount of training and testing clients selected per round.	38
IV.2	Client learning rates η_c , requested epochs E and batch-size B	40
IV.3	Comparison of the rounds needed to achieve the fixed threshold on the different CIFAR10 settings and algorithms. Resnet model.	46
IV.4	Comparison of two fairness metrics on the FEMNIST dataset, shown as gap / variance . Lower is better. Results for D03, S09, S09D05 are similar.	51
IV.5	Comparison of two fairness metrics on the Shakespeare dataset, shown as gap / variance . Lower is better. FedProx* omitted as it did not reach the required threshold.	54
B.1	ResNet model structure (with CIFAR10 input, batch of 16). The model parameters takes around 45MB.	65
B.2	TwoCNN model structure (with CIFAR10 input, batch of 16, hidden_size=32). The model parameters takes around 7MB.	66
B.3	NextCharLSTM model structure (with Shakespeare data, batch of 8). The model parameters takes around 3MB.	67
B.4	Rank by variance in terms of full test set. Ranks are by column. Values in parenthesis correspond to the difference between the max and min test accuracy between clients. Lower is better.	67

Chapter I

Introduction

1 Federated learning

Federated learning (FL) is a method for training machine learning models on distributed data silos without requiring the central collection of data. In an typical FL system, a central server coordinates the training process by sending model updates to and receiving updates from participating clients, which are often distributed devices such as smartphones or IoT sensors. The clients perform local training on their own data, and the server aggregates the updates to produce a global model. This approach addresses privacy concerns by allowing users to keep their data in place, while also addressing the practical challenges of collecting and transferring large amounts of data. It was first formally defined in the paper “Federated Optimization: Distributed Optimization Beyond the Datacenter” by Konečný, McMahan, and Ramage (2015) as “*the challenge of learning from decentralized data, where the data defining the optimization are distributed (unevenly) over an extremely large number of nodes, but the goal remains to train a high-quality centralized model*”.

Motivation Federated learning has several key benefits. First, it allows for privacy-preserving machine learning, as the raw data remains on the devices and is not shared with the central server or other devices. This can be particularly important when the data is sensitive or belongs to multiple parties with different data protection requirements. Second, federated learning can facilitate machine learning on decentralized data, which may be distributed across multiple devices or locations, and may be too large or complex to be centralized or pooled.

FL can also provide competitive advantages for organizations that are able to effectively implement it. By leveraging distributed data silos, organizations can train models on a larger and more diverse dataset than it would be possible with a centralized approach. This can result in improved model performance and increased value for the organization. Additionally, FL can enable organizations to collaborate

on model training without revealing their proprietary data, allowing them to maintain a competitive edge.

Another factor motivating the adoption of federated learning is the increasing computational capabilities of modern smartphones. The Apple A16 Bionic chip, which powers the iPhone 14, is a prime example of this trend. This chip features a 6-core CPU, 6GB of RAM, and 5-core GPU which is designed to handle demanding tasks such as gaming and augmented reality. Furthermore, the A16 Bionic includes a 16-core Neural Engine (NPU), which is capable of performing 17 trillion operations per second (TOPS), making it ideal for machine learning tasks. With such advanced hardware, smartphones are becoming increasingly capable of supporting complex machine learning algorithms locally, thereby making federated learning a viable and attractive option.

Applications In the domain of personalized recommendations, Google was among the pioneers in exploring FL for applications such as the GBoard mobile keyboard for predicting words or emojis as users type on their devices [17, 54] and search-bar suggestions in Chrome. Other major technology companies, including Apple and Microsoft, have also started leveraging FL in their products for telemetry data.

Federated Learning can be employed in the transportation industry to enhance the accuracy of traffic prediction models [42]. By training the model on data from multiple sources, such as road sensors and vehicles, FL can improve the model's ability to capture complex traffic flow patterns. This has various applications, including optimizing delivery truck routes and predicting congestion levels on roads. FL also holds potential applications in the finance industry, particularly in fraud detection and multi-party borrowing without exposing individual banks' user lists. By training a model on data from multiple financial institutions (combined with encryption mechanisms), they can better protect their customers and reduce losses due to fraud.

FL has the potential to revolutionize the healthcare field according to some recent researches [7, 11, 13, 53], where data privacy and data insufficiency have been significant barriers to developing machine learning models. By facilitating the sharing of medical data across institutions through FL, it is possible to unlock the utilization of machine learning models in the healthcare industry.

The field of natural language processing, which is particularly trendy, can also take advantage of federated learning, where models for sentiment analysis, language translation, or text summarization could be trained on diverse data sources without exposing individual users' texts. This approach ensures user privacy while enabling the development of more robust and accurate language models that can be applied across a wide range of applications and industries.

Limitations Federated learning, despite its numerous advantages, is not without challenges and limitations. One such challenge is communication efficiency, as the central server and the clients must exchange model updates and other information during the training process [18]. This can lead to increased latency and energy consumption, especially in large-scale deployments with many devices [5].

Another challenge is model convergence, as the global model may not always converge due to the decentralized nature of the data and the limited communication between the devices. Heterogeneity in data distribution and device capabilities can further exacerbate this issue [50]. Moreover, federated learning may not be suitable for certain machine learning tasks that necessitate centralized data or real-time model updates. Situations where the data is sparse, highly imbalanced, or the model requires frequent updates can limit the applicability of federated learning.

2 Research

The current thesis aims to explore key concepts in the realm of Federated Learning. The endeavor to understand and apply this complex paradigm has led us to concentrate this thesis on:

- understanding the challenges of distributed and federated optimization ;
- describing and understanding algorithmic procedures and techniques that are popular in the field ;
- applying and benchmarking popular techniques of FL on controlled data.

3 Organization

The organization follows our three research points, in order.

In the chapter II, we provide an overview of the federated learning paradigm. This includes a formal definition, a description of the process and the system and infrastructure required to support it. We also discuss and try to understand the challenges and limitations of federated learning, and outline some of the potential threats to the approach.

The chapter III focuses on the algorithmic aspects of federated learning. We begin by introducing some of the well-known algorithms used in federated learning, and provide a brief theoretical motivation. This chapter includes the algorithms of FedAvg, FedProx and FedNova which have been widely known in the field.

The chapter IV starts with a description of the setup for our experiments, including the benchmark datasets and the environment used to build, train and to evaluate the performance of the algorithms. We then present the results of our experiments and discuss the insights that we gained from them. This includes an analysis of the relative performance, robustness and fairness of the algorithms in various heterogeneous scenarios.

We finalize this work with a concluding remark and our perspective on the technique.

Chapter II

Overview of Federated Learning

Understand the paradigm

1 Definitions

Prior to the establishment of the term, technical knowledge surrounding federated learning had already appeared in scientific literature under the guise of distributed learning. Works such as [20, 27, 43] aimed to integrate separately learned models while minimizing communication costs through various techniques.

However, certain aspects were not considered significant until the publication of “Federated Optimization: Distributed Optimization Beyond the Datacenter” by Konečný, McMahan, and Ramage in 2015, which introduced the first distinction between standard distributed optimization and federated optimization by enumerating four specific characteristics: data are massively distributed across a large number of nodes, non-iid, unbalanced, and the communication is limited. To grasp a bit more the concept of federated learning, several definitions are briefly discussed.

Federated learning (FL) denotes a distributed learning paradigm that facilitates training on a substantial corpus of decentralized data unevenly distributed over an extensive number of nodes, such as mobile phones. FL essentially embodies the strategy of “*bringing the code to the data, instead of the data to the code*” (Bonawitz et al., 2019).

“Communication-Efficient Learning of Deep Networks from Decentralized Data” by McMahan et al. [35] is the paper acknowledged to have coined the term, and their inspiration for it stems from the fact that “*the learning task is solved by a loose federation of participating devices which are coordinated by a central server.*” Nevertheless, this definition somewhat restricts the scope of federated learning. The idea of training a model without data collection can be applied to cases beyond edge devices and IoT, encompassing organizations like hospitals or cutting-edge companies where data collection is undesirable or impossible due to economic, time, legal, or security constraints.

In light of this, Kairouz et al. (2021) proposed a more comprehensive definition:

“F”ederated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client’s raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective.

An additional perspective is offered by Kontar et al. (2021), which highlights that “*a practical federated learning system has the following constraint : given an evaluation metric such as accuracy, the performance of the model learning by federated learning should be better than the model learning by local training with the same model architecture*”. This way of thinking underscores the dual imperatives of Federated Learning: while the protection of data is a crucial objective, the ultimate utility and efficacy of the resultant model cannot be compromised.

Also, it might be important to distinguish between the terms **Federated Optimization** and **Federated Learning**, as they are sometimes used interchangeably despite their subtle differences in meaning. Specifically, federated optimization refers to the implicit optimization problem, or algorithmic task solved by the federated learning approach, and according to Li et al. (2018) possesses two distinct properties that substantially differentiate it from the conventional distributed setting: (1) significant variability in terms of the system characteristics on each device in the network (system heterogeneity), and (2) non-identically distributed data across the network (statistical heterogeneity).

Federated learning on the other hand is a more comprehensive framework that addresses the multitude of considerations inherent to the design and implementation of a distributed machine learning system, including network topology, data privacy, and other relevant aspects. Federated learning’s challenges are inherently interdisciplinary, and necessitates contributions and techniques from a variety of fields, including distributed optimization, cryptography, networks, and more [22]. The term “Federated Learning” thus serves as a useful shorthand, or potent terminology that encapsulates the unique traits, constraints, and challenges associated with this innovative learning paradigm.

2 Components

At its most basic, a federated learning system is composed of two critical parts: the nodes (clients) and the coordinator (server or manager).

2.1 Nodes

The nodes represent the entities that own and often generate the data. They serve as the nerve center of federated learning, embodying the most significant constraints and challenges, and ultimately driving how the entire process unfolds. The configuration and characteristics of the nodes are critical considerations: how many clients are available? What are their computational and communication capabilities and reliability? How is data related amongst the nodes? Is the data of sufficient quality? These questions are pivotal in the design and implementation of a federated learning system. Furthermore, the nodes are also the primary beneficiaries of the federated learning process, as the knowledge gained is intended to enhance the experience of these very entities.

To start with, we can discern between two types of scenarios based on the nature of the clients. One scenario involves edge devices such as smartphones, tablets, or Internet of Things devices, while the other involves organizational entities like businesses, institutions, or research laboratories. Kairouz et al. [22], from whom a definition was borrowed in the first section, introduced two terminologies to delineate these cases: **cross-device** refers to scenarios where the clients are unreliable and stateless edge devices, and **cross-silo** refers to situations where dedicated data centers act as clients. These data centers are fewer in number but are robust, consistent, and reliable. The latter, cross-silo FL, becomes particularly pertinent when legal constraints prevent a single company from centralizing its data, or when organizations with allied objectives aim to collectively enhance their models through collaboration. This could encompass diverse use cases, such as various banks collaboratively training classification or anomaly detection models to bolster fraud detection, or multiple hospitals pooling their resources to refine diagnostic models, among others.

Table II.1 summarizes the main differences between cross-silo and cross-device configurations. Note that the number of federated learning configurations that could exist is likely as varied as the range of scenarios and problems to be addressed. In this work, we present the most commonly adopted configuration as a basis for our discussions and experiments.

	Datacenter distributed learning	Cross-silo federated learning	Cross-device federated learning
Objective	Training a model on a large but "flat" dataset. Clients are compute nodes (cluster).	Training a model on a siloed data. Clients are organizations or geo-distributed data-centers.	Training a model on data spread over a very large number of nodes such as mobile or IoT devices.
Data distribution	Data is centrally stored and can be shuffled and balanced across clients.	Each client stores its own private data and cannot read other clients data. Data is often generated on the device and not iid.	
Orchestration	Centrally orchestrated	A central orchestration server organizes the training, but never sees raw data.	
Data availability	All clients are almost always available.		Only a fraction of clients are available at a given time. Often geo-correlated.
Scale	Typically less than a thousand clients.	Typically less than a hundred clients.	Typically more than a hundred, up to billions of clients.
Primary bottleneck	Computation is more often the bottleneck as datacenter are assumed to have very fast networks.	Computation and/or communication.	Communication is the often primary bottleneck, though it depends on the task.
Addressability	Stateful. Each client has an identity that allows the system to access it specifically.		Stateless. Clients cannot be indexed or identified.

Table II.1: Adapted from Table 1 of Li et al. (2020), compares cross-device and cross-silo systems with the traditional datacenter distributed learning across various dimensions. These characteristics play a critical role in determining which algorithms and techniques are suitable for a given scenario.

Then, depending on how the data is distributed across the feature and sample spaces, we categorize the setting as either Horizontal Federated Learning (HFL) or Vertical Federated Learning (VFL) [52]. While horizontally partitioned data is commonly encountered, VFL is less prevalent except in cross-silo scenarios.

In HFL, several parties possess similar features but little to no overlapping samples. This configuration is the most commonly encountered in the literature, and it allows for the utilization of the same model structure across all clients. VFL is of application when different datasets share an identical sample space but encompass distinct feature spaces. Here, each client has unique features, and their collaborative efforts result in a more comprehensive feature landscape for the shared sample IDs. In VFL, aligning (or matching) observations across different features is necessary and requires additional techniques such as Private Set Intersection (PSI) [14, 16]. Figure II.1 illustrates the nuance.

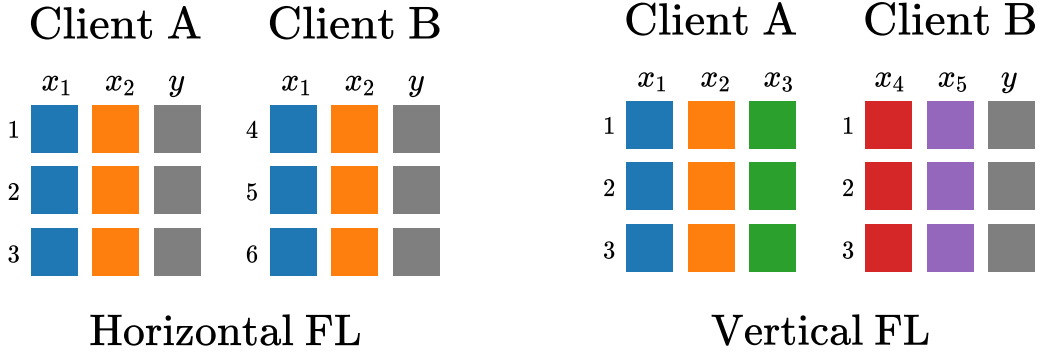


Figure II.1: Visual illustration of horizontal FL and vertical FL. Columns are the features and labels spaces $\{\mathcal{X}, \mathcal{Y}\}$. Rows represents clients samples, labelled by IDs for visual understanding and comparison purposes.

2.2 Manager

The classic federated learning ecosystem is orchestrated by an entity known as the manager, or coordinator, which plays a pivotal role in the establishment and maintenance of the global machine learning model. In a cross-device context, the manager's role is typically filled by a robust central server and is of paramount importance to ensure the stability and reliability of the system. The manager is responsible for coordinating communication between participants executing the global training process.

In a cross-silo setting, the manager role can be assumed by one of the participating organizations. This is especially common in vertical federated learning (VFL) contexts, where only one party may possess the labels [33]. Here, the manager is accountable for aligning and aggregating data from different participants, orchestrating secure

communication channels, and facilitating collaborative model building.

The major role of the manager is not without its challenges. As the number of clients grows, the manager can become a communication bottleneck, impeding the smooth flow of information and the efficiency of the learning process. Moreover, the centralized nature of the manager presents a potential vulnerability in the system by constituting a single point of failure. These potential issues have prompted further exploration of fully decentralized solutions, which distribute the responsibilities and workload of the manager across the network, fostering a more resilient and scalable federated learning system.

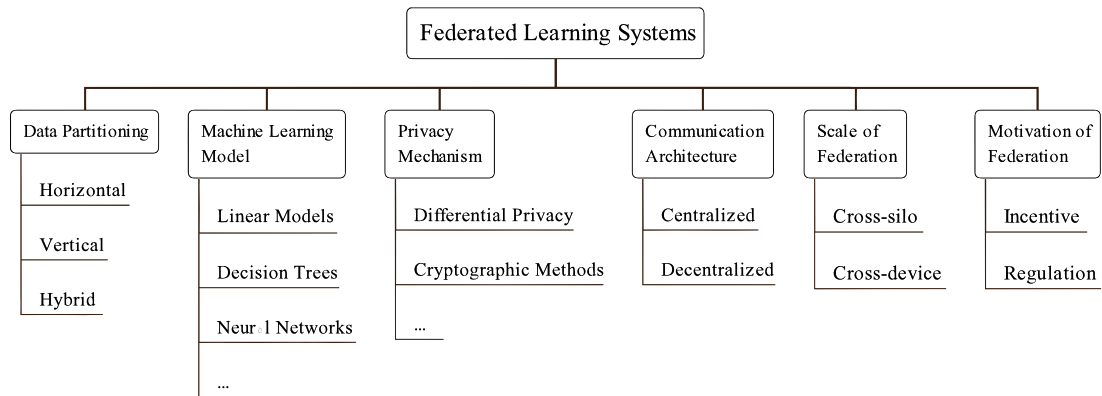


Figure II.2: Taxonomy of FL Systems, by Li et al. (2021) [28]

3 Process

In this section, we outline the standard procedure typically employed in Federated Learning. This iterative (also synchronous and cyclical) process continues until a specified stopping condition is met, such as a maximum number of rounds, a desired level of model performance, or an acceptable level of model convergence. It's important to note that, once more, this described process is a typical representation of cross-device and horizontal federated learning without any added mechanism, and the exact process can vary depending on specific use cases and requirements.

Figure II.3 offers a simplified depiction of the training and aggregation stages of Federated Learning. In the illustrated scenario, five devices are selected to partake in the learning process. The initial model is broadcasted to these devices, each of which conducts local training using its own data. Subsequently, they transmit their locally trained models back to the server, with the exception of the black client dropping out (for diverse reasons such as loss of internet connectivity). Finally, the server consolidates all the received updates to generate a new, combined model.

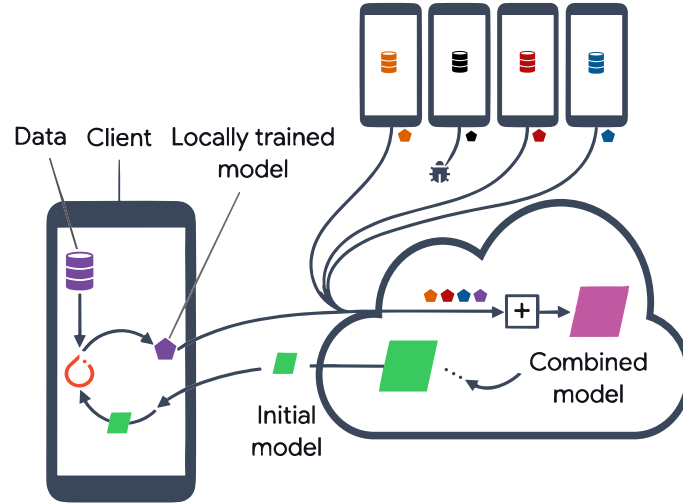


Figure II.3: Simplified illustration of the training process (cross-device).

Vectorized, then modified from *What is Federated Learning* [46]

1. Initialization phase :

- (a) Client selection : the clients, based on several factors mark themselves as available. The first step of the server involves choosing (a subset of) available clients to participate in the current round of model training.
- (b) (Global model) : for the first round, the initialization of the global model can either be done using centralized training data if available to the server or at random.
- (c) Broadcasting : the selected clients are sent the global model parameters. This action initiates the process of training on each client's local data.

2. Training phase :

based on a defined strategy, each client computes an updated model based on its local data. This step is accomplished by running one or several iterations of a specified machine learning algorithm. The computation is done locally on each client.

3. Aggregation phase :

- (a) Aggregation : upon receipt of the local updates, the server proceeds to aggregate them, typically through a form of averaging to generate a global model. Subsequently, this updated global model serves as the basis for the commencement of the next federated learning round.
- (b) (Evaluation) : the evaluation of the global model can be conducted on centrally available data, or alternatively, on a subset of clients using their local data.

4 Challenges

4.1 Statistical heterogeneity

One of the prominent challenges in FL stems from the wide-ranging and diverse nature of the data utilized for model training. Suppose $\mathcal{D}_c$ represents the undisclosed local data distribution of a specific client c . Non-IIDness typically designates the discrepancies between $\mathcal{D}_i$ and $\mathcal{D}_j$ for any two clients i and j , given that $i \neq j$. As stated by Kairouz et al. (2021), there are several usual manners in which data deviate from being identically distributed.

- **Feature distribution skew:** also known as covariate shift, this pertains to scenarios where the feature distribution varies across clients. An instance of this could be observed in handwriting recognition tasks, where different individuals may write the same digits/letters in unique manners (differing stroke widths, slants, etc).
- **Label distribution skew:** variations in label distributions are prevalent when data originates from diverse users or locations. An illustration of this could be an image classification task, where images of tulips are likely to be more prevalent in the data of Dutch clients than others.
- **Label-features inconsistency:** different clients might associate the same label with distinct features due to cultural, geographical, or other factors. For instance, images categorized as “fruit” might depict different varieties such as apples, oranges, and bananas, with varying prominence depending on the preferences and local availability of fruits in different regions.
- **Features-label inconsistency:** the same features could be associated with different labels depending on the client. For example, in sentiment analysis of text, an identical sentence like “Oh, another original superhero movie... love it!” might evoke positive sentiment from some clients, while others might interpret it as sarcasm or express a negative sentiment based on their individual perspectives or cultural nuances.
- **Class imbalance:** this refers to the situation when certain classes within the data have more examples than others. This could lead to a model that is biased towards the overrepresented classes, thereby hindering its ability to accurately predict the underrepresented ones.

4.2 System heterogeneity

Cross-device federated learning involves a broad spectrum of clients, each with distinct computational and communication abilities. This diversity presents substantial hurdles in achieving satisfactory model performance and convergence. Particularly, clients with limited resources, such as mobile devices, may find it challenging to perform the necessary computations for model training in a timely manner.

Stragglers are clients that require more time to complete their computations compared to others. This is often the case with clients that have limited computational resources, such as mobile devices, which may struggle to perform the necessary computations for model training promptly. The existence of stragglers can considerably decelerate the overall learning process. This is particularly noticeable in synchronous federated learning settings, where the server waits for all participating clients to provide updates before advancing to the next training round. If we denote T_c the time required for local computation at a specified client c the total time the server must wait without excluding stragglers is $\max(T_1, T_2, \dots, T_C)$, which could be significantly long.

One strategy to mitigate the impact of stragglers is to interact with a smaller proportion $K < 1$ of clients per round. This reduces in expectation the total wait time per round, as $\max(T_1, T_2, \dots, T_C) \geq \max(T_1, T_2, \dots, T_{C \times K})$. However, this approach might slow down the convergence speed since fewer updates are made to the global model in each round. It also risks introducing bias into the model if the selected clients do not represent the overall data distribution accurately.

Another approach to handle stragglers is the deadline-based approach, where the server sets a fixed maximum computation time [38]. At the end of this period, clients send their updates, regardless of whether the number of iterations matches the requested number. This approach is the one chosen in our experiments.

Finally, the central server can use adaptive sampling or importance sampling techniques to selectively choose which clients to communicate with during each iteration. This achieves a balance between convergence speed and communication overhead, providing a more efficient way to handle stragglers. An example of this is “Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity” [41].

Dropouts refer to the clients that become unavailable or fail to participate in the learning process at certain times. Client dropout can occur due to various reasons, including network disconnections, low bandwidth, device shutdowns, or simply because the user chooses to opt out of the learning process. Dropouts can

lead to inconsistencies in the learning process, as the server may not receive updates from all clients in each round of training. This can add unwanted biases that affect the accuracy and robustness of the global model, as it may not fully capture the knowledge from the the weakest clients.

4.3 Evaluation and monitoring

The decentralized nature of federated learning systems and the stringent requirements for data privacy present unique challenges for monitoring the training process and evaluating the model’s performance. Two key challenges are the inability to access raw data at each client and the acquisition of precise ground truth labels [10].

Therefore, monitoring the training process in a federated learning context requires a thoughtful approach that acknowledges the decentralized nature of the system and the need for data privacy. Two strategies are centralized evaluation and federated evaluation (Beutel et al., 2020).

Centralized evaluation also known as server-side evaluation, is a straightforward approach that leverages a centrally stored dataset to evaluate the model. The advantage of this approach is the consistent accessibility of the entire evaluation dataset, which enables consistent model evaluation. However, the feasibility of this technique depends on the existence of such a dataset, which may be challenging or even unachievable to obtain for certain specific tasks.

Federated evaluation or client-side evaluation, involves assessing the model via the clients, leading a more authentic evaluation of the model’s performance. This approach is commonly preferred in real-world scenarios. Two prevalent strategies are reserving a portion of each client’s data for evaluation, or designating a subset of clients as a test set in each round. However, the inherently dynamic composition of the evaluation datasets across learning rounds, coupled with the potential for changes in each client’s dataset over time can lead to unstable evaluation outcomes, even if the model itself remains unchanged.

4.4 Threats

Privacy is one of the fundamental concerns in federated learning, as it involves the participation of multiple parties who may have sensitive data that they do not want to reveal to others. However, federated learning is not immune by nature to security threats and may be susceptible to attacks. One false assumption of federated learning is that participating devices are trustworthy and do not manipulate or steal data parameters. In reality, some devices or machines may be untrusted or adversarial, which can compromise the security or privacy of the federated learning system. In

fact, even honest clients may unintentionally become adversarial due to compromised devices, software bugs or hardware faults, data quality issues, or model drift in dynamic environments where the data distribution changes over time.

Lyu et al. (2020) provides an extensive overview of potential threats to federated learning. Broadly, these threats fall into two categories: poisoning attacks and inference attacks.

Poisoning attacks encompass malicious actions aimed at degrading the performance or functionality of the global model. These attacks manifest in two primary forms:

- **Data poisoning:** in this scenario, adversaries manipulate their local data whether by altering, adding, or deleting samples to negatively influence the global model. For example, compromised devices might introduce corrupted training samples with flipped or noisy labels [29].
- **Model poisoning:** instead of tampering with data, attackers manipulate the local model updates directly. The goal can be generic, such as preventing accurate model training, or specific, like embedding backdoors to trigger certain behaviors in the global model. Such targeted manipulations are detailed in Dai and Li (2023).

Inference attacks represent the other significant category of threats and aim to extract sensitive information from the global model or participating clients. These attacks can be initiated by both insiders and outsiders and fall into two fundamental categories: membership inference attacks, and reconstruction attacks.

- **Reconstruction :** the aim here is to deduce attributes of training records. By exploiting nuances in the global model or model updates, attackers can approximate or even recover original data, which could reveal sensitive client information. For instance, techniques like Deep Leakage from Gradients (DLG) detailed by Zhu, Liu, and Han (2019), and its subsequent enhancements Zhao, Mopuri, and Bilen (2020), demonstrate the feasibility of such attacks on any differentiable model trained using cross-entropy loss over one-hot labels. Further, tabular data which are frequently used in high-stakes applications are not exempt of leakage, as demonstrated by Vero et al. (2022).
- **Membership inference:** These attacks try to ascertain if a specific data point was in a client’s local dataset by observing global model behavior. As Nasr, Shokri, and Houmansadr (2019) suggests, adversaries can exploit the learning process itself, such as by manipulating gradient updates, to glean information about data membership.

It's also crucial to note that while clients can pose risks, the server itself might act maliciously. The server, having access to aggregated updates, can exploit this data, compromising the privacy of individual clients. These threats underscore the absolute necessity for robust security mechanisms in a production environment.

4.5 Protections

In the quest to ensure the integrity and privacy of federated learning, differential privacy and secure aggregation emerge as first lines of defense against vulnerabilities.

Differential Privacy is a concept that revolves around limiting the contribution of a single client and adding a touch of randomness to the data. By doing so, it makes sure that a single client's data addition or removal doesn't sway the result significantly. Such obfuscation makes it difficult for the server to execute specific attacks or infer details about a client's data [36, 1]. However, the addition of noise can have a significant impact on the utility of the updates, and degrade convergence. We'll dive deeper into an algorithm leveraging differential privacy with adaptive clipping in the upcoming chapter. Refer to subsection 5.1 for more details.

Secure Aggregation is a technique that ensures the model updates are encrypted in such a way that only the cumulative updates from all clients can be decrypted by the server, not the individual ones. To bolster user privacy, renowned secure aggregation protocols like SecAgg [6] and SecAgg+ (Bell et al., 2020) utilize secret keys and seed generated at the client end. This masks their updates, and the corresponding public key is then shared with the server. This setup facilitates the reconstruction of the aggregate by nullifying the masks. But there's a catch. The intricacy of these methods multiplies with the increase in user count, especially when users drop out. This is because the missing key segments need to be pieced back together, which is a challenging task.

Multiple efforts have been made to amplify the efficiency of secure aggregation. For instance, TurboAgg (So, Guler, and Avestimehr, 2020) adopts a circular communication pattern to reduce the communication burden. However, it adds to round complexity and offers a diluted privacy guarantee in comparison to SecAgg since it only ensures model privacy in an average situation rather than the most unfavorable scenario. Meanwhile, FastSecAgg (Kadhe et al., 2020) optimizes per-user overhead through the Fast Fourier Transform multi-secret sharing. Yet, it also slightly compromises on privacy and dropout guarantee. A recent development in this domain is the LightSecureAggregation protocol (So et al., 2021). It addresses the reconstruction bottleneck by shifting from the conventional "random-seed reconstruction" to a novel "one-shot aggregate-mask reconstruction" approach.

Chapter III

Methods and Algorithms

Litterature review

1 Problem formulation

Federated learning, in its conventional understanding, aims to collaboratively train a singular, global statistical model. This is done under the constraint that data is partitioned and stored across multiple nodes that are unable to communicate directly with each other, and the data is not publicly accessible.

Definition 1 (Cross-device Problem Formulation).

- Let ζ denote the collection of all possible clients, which may be infinite.
- Each client $c \in \zeta$ possesses its own private data collection $D_c = \xi_1, \dots, \xi_{n_c}$ of size n_c , which follows the local data distribution $\mathcal{D}_c$.
- The sample $\xi \in D_c$ is an input-output (features-label) pair $\{x, y\} \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ represents the feature space and $\mathcal{Y}$ represents the label space.
- Furthermore, we assume a (meta-)distribution $\mathcal{C}$ supported on ζ .

The goal is to optimize the following objective [49, 55]:

$$F(\mathbf{w}) \stackrel{\text{def}}{=} \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{\xi \sim \mathcal{D}_c} [f_c(\xi, \mathbf{w})] \right], \quad (\text{III.1})$$

This objective can be decomposed into two parts: $F(\mathbf{w})$, the overall objective function, and $F_c(\mathbf{w})$, the local objective function for device c with model parameters $\mathbf{w}$. The decomposition is as follows:

$$F(\mathbf{w}) = \mathbb{E}_{c \sim \mathcal{C}} [F_c(\mathbf{w})] \quad \text{where} \quad F_c(\mathbf{w}) = \mathbb{E}_{\xi \sim \mathcal{D}_c} [f_c(\xi, \mathbf{w})]. \quad (\text{III.2})$$

While the same loss function $f_c(\xi, \mathbf{w})$ is commonly considered across the clients, the local objectives $F_c(\mathbf{w})$ typically vary significantly due to the variability in the data distributions $\mathcal{D}_c$ (see subsection 4.1).

The formulation of III.2 is particularly justified in the cross-device setting, where the number of clients is massively large. However, while the goal is to optimize the expected global loss over the client local distributions, at a given time, only a finite sample of C clients is available for training. We can rewrite III.2 in the form of a combination of the local objective available locally to node c , i.e., an empirical risk minimization objective function (ERM), with a finite set of clients and where each client has finite local data points [49].

$$F(\mathbf{w}) = \sum_{c=1}^C p_c F_c(\mathbf{w}) \quad \text{where} \quad F_c(\mathbf{w}) = \frac{1}{n_c} \sum_{\xi \in D_c} f_c(\xi, \mathbf{w}). \quad (\text{III.3})$$

Here, the client weighting parameter p_c can be freely chosen and denotes the relative significance attributed to each client. Two commonly employed configurations for this parameter are:

$$p_c = \frac{1}{C} \quad \text{or} \quad p_c = \frac{n_c}{\sum_{c=1}^C n_c}. \quad (\text{III.4})$$

The latter configuration transforms equation (III.3) into the empirical risk minimization (ERM) of the union of all the local datasets $D \stackrel{\text{def}}{=} \bigcup_{c=1}^C D_c$, as per Wang et al. [49]. Empirical evidence suggests that this configuration often offers superior performance when compared to uniform weighting, as documented by [56, 40].

2 Solving the optimization problem

This section presents an algorithmic framework for federated optimization, as proposed by Reddi et al. in their seminal work “Adaptive Federated Optimization”. This framework is designed to address the unique challenges of optimizing machine learning models in a decentralized and distributed setting. It provides a unified representation for a multitude of algorithms. We begin by discussing the primary algorithm, which offers the most intuitive approach to solving the optimization problem delineated in section 1.

2.1 FedOpt : the framework

The FedOpt framework adheres to the "computation then aggregation" (CTA) protocol. It aggregates stochastic gradients computed locally by the clients and uses

them to update a global model. The framework is designed to be compatible with gradient-based optimizers.

Let's define T as the number of communication rounds to be performed, E_c as the number of local epochs we wish client c to train for, and p_c as the client weight (as seen in equation III.4). Additionally, let η_c and η_s be the learning rates for the client and server optimizers, respectively. We use the notation $(*)$ to denote additional arguments that are left to the specific needs of algorithms.

Algorithm 1 Generalized Federated Optimization (FedOpt)

Input functions : CLIENTOPT, SERVEROPT, AGGREGATION

Parameters : $T, E_c, p_c, \eta_c, \eta_s, *$

```

1 Initialize global model  $\mathbf{w}^0$ 
2 for each round  $t = 0, 1, \dots, T - 1$  do
3   Sample a subset  $S^t$  of clients
4   for each client  $c \in S^t$  in parallel do
5     

---


6      $(\Delta_c^t, *) = \text{CLIENTOPT}(\mathbf{w}^t, \eta_c, *)$ 
7     return  $(\Delta_c^t, *)$  to the server
8     

---


9   end for
10   $\Delta^t = \text{AGGREGATION}(S^t, \Delta_c^t, p_c, *)$ 
11   $\mathbf{w}^{t+1} = \text{SERVEROPT}(\mathbf{w}^t, \Delta^t, \eta_s, *)$ 
12 end for
```

The FedOpt framework is highly flexible, allowing for customization of various aspects such as the client update rule, the global model update rule, and the aggregation method used for updates. This algorithm is parameterized by two gradient-based optimizers, CLIENTOPT and SERVEROPT, and an additional AGGREGATION function at the server level.

In its simplest form, one can use SGD for both SERVEROPT and CLIENTOPT optimizers, and use fixed learning rates with the same number of iterations across clients. By setting $\eta_s = 1$, and constraining η_c to be fixed during rounds, we obtain a well-known algorithm: Federated Averaging.

2.2 FedAvg : the baseline

Federated Averaging McMahan et al. [35] is a simple yet effective algorithm for federated optimization that has become a widely-used benchmark for federated learning algorithms. Its simplicity and interpretability make it an ideal starting point for exploring various federated learning applications.

The development of the Federated Averaging algorithm, as presented in the original paper, begins with a simple idea: optimize each client's local objective function using stochastic gradient descent. If each client computes $\nabla F_c(\mathbf{w})$, the central server then applies the update:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_s \sum_{c \in S^t} p_c \nabla F_c(\mathbf{w}^t). \quad (\text{III.5})$$

Since the sum of the weighted gradients approximates the gradient of the global objective, an equivalent update is updating the global model weights as the mean of the local weights:

$$\mathbf{w}^{t+1} = \eta_s \sum_{c \in S^t} p_c \mathbf{w}_c^t \quad \text{where} \quad \mathbf{w}_c^t = \mathbf{w}^t - \nabla F_c(\mathbf{w}^t) \quad (\text{III.6})$$

By adding more computation to each client and iterating the local update $\mathbf{w}_c^{t,e+1} = \mathbf{w}_c^t - \nabla F_c(\mathbf{w}_c^{t,e})$ for, say, E times (notice the use of $\mathbf{w}_c^{t,e}$ in the gradient computation), we obtain FedAvg. To align with the way updates are handled by the FedOpt framework [40], we need to make another transformation. Instead of updating the global model directly with the local model parameters, we retrieve only the local update (i.e., the difference between the global model and the trained local model parameters). This is equivalent but emphasizes the use of (pseudo) gradient descent at the server level. In fact, by rewriting the update rule as

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_s \sum_{c \in S^t} p_c (\mathbf{w}^t - \mathbf{w}_c^{t,E}), \quad (\text{III.7})$$

and by defining $\Delta_c^t = (\mathbf{w}_c^{t,E} - \mathbf{w}^t)$, and $\Delta^t = \sum_{c \in S^t} p_c \Delta_c^t$, we can find the global update rule (SERVEROPT) present in line 15 of Algorithm 2.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_s \sum_{c \in S^t} p_c (-\Delta_c^t) = \mathbf{w}^t + \eta_s \Delta^t. \quad (\text{III.8})$$

Implementation The algorithm starts by initializing the global parameters, either randomly or using a pretrained model. It then conducts synchronous rounds of communication as follows:

During each round t , the server selects a subset of clients by taking a fraction of clients $S_t = C \cdot K$ with $K > 0$ and dispatches the current global model $\mathbf{w}^t$ to all of them. Each client $c \in S^t$ updates its local model with the received shared model, setting $\mathbf{w}_c^{t,0} = \mathbf{w}^t$. The client then divides its local training data into batches of size B and carries out E epochs of (stochastic) gradient descent to compute the local update Δ_c^t . The server collects the local updates from every chosen client and generates the new global model by computing a weighted average of all the local updates. This new global model serves as the starting point for the next communication round. These steps are iteratively repeated for T rounds.

Algorithm 2 FedAvg

Functions of the framework

```

1  CLIENTOPT(*):
2     $\mathbf{w}_c^{(t,0)} = \mathbf{w}^t$ 
3    for epoch  $e = 0, 1, \dots, E_c$  do
4      for each batch  $\mathcal{B}$  of size  $B$  do
5         $\mathbf{g}_c^t(\mathcal{B}) = \frac{1}{B} \sum_{\xi \in \mathcal{B}} \partial_{\mathbf{w}} f_c(\xi, \mathbf{w}^{(t,e)})$  ▷ Minibatch-SGD
6         $\mathbf{w}_c^{(t,e+1)} = \mathbf{w}_c^{(t,e)} - \eta_c \mathbf{g}_c^t(\mathcal{B})$ 
7      end for
8    end for
9    return  $\mathbf{w}_c^{(t,E_c)} - \mathbf{w}^t$ 
10
11 AGGREGATION(*):
12   return  $\sum_{c \in S^t} p_c \Delta_c^t$  ▷  $p_c : \frac{n_c}{\sum_{c \in S^t} n_c}$ 
13
14 SERVEROPT(*):
15   return  $\mathbf{w}^t + \eta_s \Delta^t$ 

```

Effects of the parameters

An aspect to consider is the number of local iterations on the data, determined by parameters E and B . These parameters directly influence the total computation required. Each client performs E passes over its local dataset, using a mini-batch size of B at every iteration. The total amount of local computation for each client is given by $\tau_c = E \cdot n_c / B$, which can be increased by either reducing B or raising E .

According to the authors' studies, as long as the batch size B is substantial enough to harness the full potential of device parallelism, there is no incurred cost in reducing the batch size B . A rise in local computations can lead to lower communication costs for reaching a specific accuracy level. This effect is illustrated in Figure III.1. The left figure represents two communication rounds with clients undertaking a single pass through their data, while the right figure portrays one communication round with several local epochs.

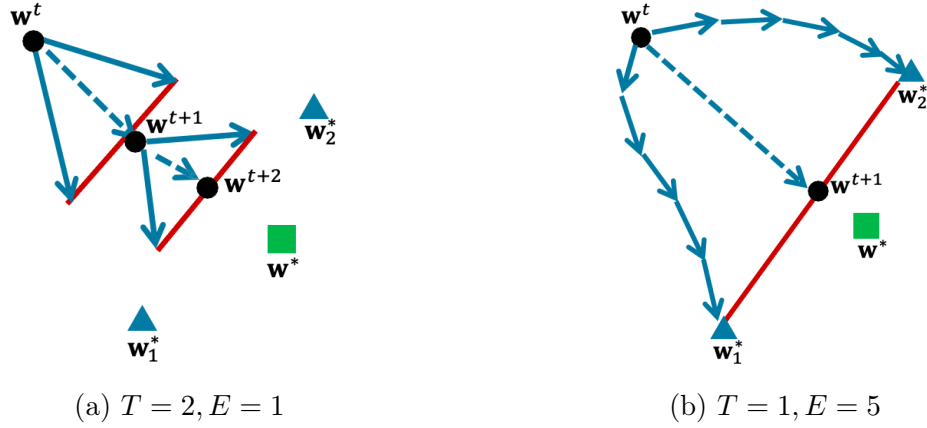


Figure III.1: A comparison of the number of local epochs for the FedAvg algorithm with two clients. The local updates (depicted by solid blue arrows) gradually converge towards the local optima of their respective clients (indicated by blue triangles). The server update (dashed blue arrows) moves towards the average of the local optima of the selected clients $\sum_{c \in S^t} p_c \mathbf{w}_c^*$. The global optimum is represented as a green rectangle. Not accurately drawn.

Another parameter that can reduce the number of global-level exchanges needed to achieve a fixed accuracy is the ratio C of selected clients. By selecting more clients, one can leverage more training data, leading to convergence in fewer rounds of communication. However, it is important to strike a balance between selecting too few or too many clients. Synchronous algorithms like FedAvg can face challenges when selecting a large number of clients. As seen in section subsection 4.2, these algorithms follow a “computation then aggregation” (CTA) approach, requiring all selected clients to compute their updates before the server can aggregate them and proceed to the next communication round. Consequently, the overall runtime of the algorithm is dominated by the slowest clients. To complete a communication round, it either has to wait for the slowest clients to finish or to drop them. The latter strategy is the one implicitly used in the Federated Averaging algorithm. However, dropping slow clients can result in suboptimal convergence [31]. Conversely, selecting too few clients per round may lead to insufficient diversity and a significant increase

in stochastic noise in the updates, which can also negatively impact the convergence of the algorithm [49].

3 Tackling the heterogeneity

3.1 FedProx : proximal term

Li et al. (2018) propose the FedProx algorithm to address the challenges of federated optimization in heterogeneous networks. FedProx is presented as a generalization and re-parametrization of FedAvg, which aims to have a more stable convergence behavior and be efficient in highly heterogeneous settings. It tries to do so by focusing on systems and data heterogeneity (see subsection 4.2) with two tweaks [31] :

1. Introducing the definition of an γ -inexact solution, allowing clients to have variable number of epochs, and carefully merging (even partial) solutions accordingly. This is a theoretical add-on.
2. Introducing a proximal term to the objective function as a means to restrict the local updates made by participating devices to be closer to the initial global model. It accounts for the heterogeneity in model progress that arises when only partial information is available.

The idea of penalizing local models that significantly deviate from the global model by adding a regularization term to the local objectives naturally emerges from the observation that excessive local optimization can cause local models to converge to the minima of local objectives. Mathematically, this can be achieved by introducing a distance metric $\psi(\cdot)$ that quantifies the difference between the local model $\mathbf{w}_c^t$ and the global model $\mathbf{w}^t$ at iteration t . This distance measure is integrated into the (augmented) local objective function as follows:

$$h_c(\mathbf{w}_c^t, \mathbf{w}^t) = F_c(\mathbf{w}_c^t) + \psi_c(\mathbf{w}_c^t, \mathbf{w}^t) \quad (\text{III.9})$$

Specifically, FedProx introduces a regularization function containing a quadratic regularizer term (i.e., Euclidean distance) between the local models and the global model. This regularization function, represented as

$$\psi(\mathbf{w}_c^t, \mathbf{w}^t) = \frac{\mu}{2} \|\mathbf{w}_c^t - \mathbf{w}^t\|^2 \quad (\text{III.10})$$

acts as a *proximal term* that is included in each local objective. By restricting the model parameters to remain close to their previous values, the proximal term helps ensure that the local models do not deviate too much from the global model during the training process. Notably, this method does not incur any additional

communication cost compared to FedAvg, making it a suitable option for practical implementation in federated learning systems [31, 25, 49].

Moreover, instead of requiring each client to find the exact optimal solution to the local optimization problem, the γ -inexact solution allows clients to find a solution within a factor γ of the optimal local solution. As clients may possess non-IID data and varying resource constraints, expecting each client to find the exact optimal local solution may not be realistic. Allowing clients to find an approximate solution within a factor γ , the federated optimization algorithm can accommodate federated learning's practical constraints. Mathematically, this yields the following definition (Def. 2 from [31]):

Definition 2 (γ_c^t -inexact solution). Given a function $h_c(\mathbf{w}, \mathbf{w}^t)$ and $\gamma \in [0, 1]$, we say $\mathbf{w}^*$ is a γ_c^t -inexact solution of $\min_{\mathbf{w}} h_c(\mathbf{w}, \mathbf{w}^t)$ if $\|\nabla h_c(\mathbf{w}^*, \mathbf{w}^t)\| \leq \gamma_{t,k} \|\nabla h_k(\mathbf{w}^t; \mathbf{w}^t)\|$, where $\nabla h_c(\mathbf{w}; \mathbf{w}^t) = \nabla F_k(\mathbf{w}) + \mu(\mathbf{w} - \mathbf{w}^t)$.

Setting γ to 0 implies that local solvers on all devices must find an exact solution to their local subproblems at each iteration. In other words, the optimization process must be entirely accurate, and there is, in theory, no tolerance for inexactness.

Implementation

Remark. The notation $\tilde{\mathbf{g}}_c^t(\mathcal{B})$ is used in this context to represent the fact that any local algorithm OPTIMIZER can be used to inexactly solve h_c . In practice we typically let each device start from the current global model $\mathbf{w}^t$ and run multiple local iterations of minibatch-SGD, which is a natural way to obtain an inexact solution.

Algorithm 3 FedProx Algorithm

Functions of the framework

```

1  CLIENTOPT(*):
2     $\mathbf{w}_c^{(t,0)} = \mathbf{w}^t$ 
3    for epoch  $e = 0, 1, \dots, E_c$  do
4      for each batch  $\mathcal{B}$  of size  $B$  do
5         $h_c = F_c(\mathbf{w}_c^{(t,e)}) + \frac{\mu}{2} \|\mathbf{w}_c^{(t,e)} - \mathbf{w}^t\|^2$ 
6         $\tilde{\mathbf{g}}_c^t(\mathcal{B}) = \text{OPTIMIZER}(h_c)$ 
7         $\mathbf{w}_c^{(t,e+1)} = \mathbf{w}_c^{(t,e)} - \eta_c \tilde{\mathbf{g}}_c^t(\mathcal{B})$ 
8      end for
9    end for
10   return  $\mathbf{w}_c^{(t,E_c)} - \mathbf{w}^t$ 
11
12  AGGREGATION(*):
13   return  $\sum_{c \in S^t} p_c \Delta_c^t$ 
14
15  SERVEROPT(*):
16   return  $\mathbf{w}^t + \eta_s \Delta^t$ 

```

Effects of the parameters

The tuning of μ is of particular importance as the convergence depends on it. There is an important trade-off between the convergence and the stability. The authors say “a large μ may potentially slow the convergence by forcing the updates to be close to the starting point, while a small μ may not make any difference [to FedAvg].” [31]

While difficult to automatically tune the parameter, they propose the heuristic of adaptively increasing or decreasing μ in the same direction as the loss. Specifically, they adjust the penalty by a certain amount when the loss goes in the same direction for some consecutive rounds. We will experiment on this approach, see subsection 1.1.

3.2 FedNova : objective consistency

“While FedProx improve the local training stage of FedAvg, FedNova improves the aggregation stage of FedAvg. It takes the heterogeneous local updates of each party into consideration and normalizes the local models according to the local updates before averaging.” –Li et al. (2021)

In federated learning, the variability of data size and statistical distribution across different clients leads to heterogeneity, which poses a challenge for convergence.

Computational heterogeneity can also exacerbate this problem, where clients with larger datasets, faster processing, or different local optimizers may contribute to heterogeneous progress towards the global objective.

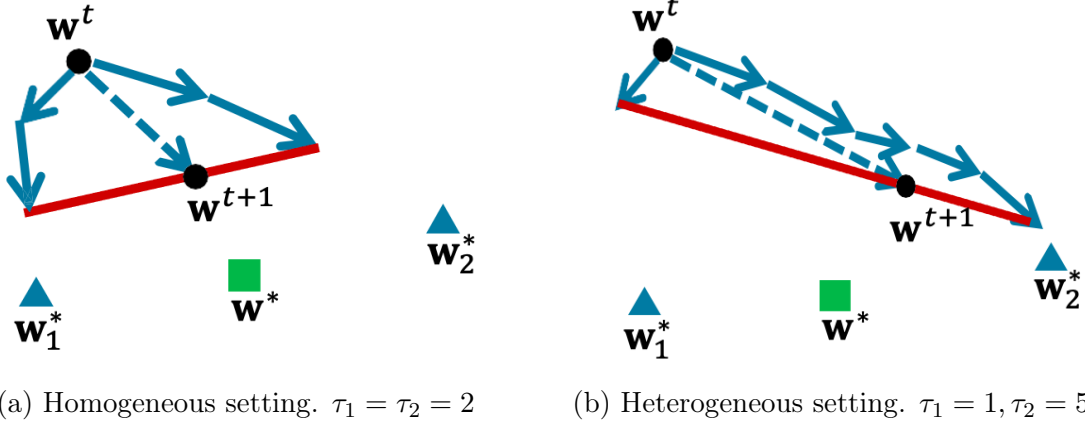


Figure III.2: Comparison of the model updates after standard averaging of 2 client models after homogeneous (a) and heterogeneous (b) local updates to illustrate the objective inconsistency. Blue triangles and green squares are the minima of local and global models while solid and dashed blue lines are the local and global updates, respectively.

This heterogeneity can give rise to a serious issue defined as *objective inconsistency*. When the number of local updates differs across clients $\tau_i \neq \tau_j, \forall i \neq j$, the updated global model can become biased towards the local objective of the client that performs more local updates. Instead of optimizing the true global objective federated learning in this heterogeneous setting ends up optimizing a mismatched objective, which can significantly deviate from the true objective. See Figure III.2b for illustration.

To address objective inconsistency Wang et al. (2020) present federated normalized averaging (FedNova) as a new generalized algorithm. To start with, they consider the accumulated update to be a linear combination of local gradients such that:

$$\Delta_c^t = -\eta \mathbf{G}_c^t \mathbf{a}_c^t, \quad (\text{III.11})$$

where $\mathbf{G}_c^t = [\mathbf{g}_c^{(t,0)}, \mathbf{g}_c^{(t,1)}, \dots, \mathbf{g}_c^{(t,\tau_c)}] \in \mathbb{R}^{d \times \tau_c}$ contains all the stochastic gradients of client c in round t , and $\mathbf{a}_c^t \in \mathbb{R}^{\tau_c}$ is a non-negative vector that depends on the local optimizers and defines how the gradients are locally accumulated. Then, based on

Equation III.11, Wang et al. derive the following formulation:

$$\begin{aligned}\Delta^t &= \sum_{c=1}^C p_c \Delta_c^t = - \sum_{c=1}^C p_c \|\mathbf{a}_c^t\|_1 \cdot \frac{\eta \mathbf{G}_c^t \mathbf{a}_c^t}{\|\mathbf{a}_c^t\|_1} \\ &= - \underbrace{\left(\sum_{c=1}^C p_c \|\mathbf{a}_c^t\|_1 \right)}_{\tau_{\text{eff}}: \text{ effective local steps}} \sum_{c=1}^C \underbrace{\eta \left(\frac{p_c \|\mathbf{a}_c^t\|_1}{\sum_{c=1}^C p_c \|\mathbf{a}_c^t\|_1} \right)}_{w_c: \text{ weight}} \underbrace{\left(\frac{\mathbf{G}_c^t \mathbf{a}_c^t}{\|\mathbf{a}_c^t\|_1} \right)}_{\mathbf{d}_c: \text{ normalized gradient}}.\end{aligned}\quad (\text{III.12})$$

Therefore, a generalized rewritten version of the federated learning update rule can be obtained, where instead of sending the local progress Δ_c^t , each client sends a normalized gradient, denoted by $\mathbf{d}_c$. Additionally, the update rule includes an extra term τ_{eff} , which denotes “the effective number of steps”, or more analogously the global learning rate. Replacing the elements by their notation yields the following update rule:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \tau_{\text{eff}}^t \sum_{c=1}^C w_c \cdot \eta \mathbf{d}_c^t \quad \text{where} \quad \mathbf{d}_c^t = \frac{\mathbf{G}_c^t \mathbf{a}_c^t}{\|\mathbf{a}_c^t\|_1}. \quad (\text{III.13})$$

This new update rule allows the authors to present two theorems. The first theorem states that under certain optimization assumptions (Unbiased Gradient (A.1), Bounded Variance (A.2), Smoothness (A.3) and Bounded Dissimilarity (A.4)), any federated optimization algorithm following the update rule specified in Equation III.13 will converge to a stationary point of the surrogate objective function:

$$\tilde{F}(\mathbf{w}) = \sum_{c=1}^C w_c F_c(\mathbf{w}). \quad (\text{III.14})$$

The second theorem posits that under the same conditions, there is a non-vanishing error due to objective inconsistency: $\epsilon = 2\chi_{\mathbf{p}||\mathbf{w}}^2 \kappa^2$, where $\chi_{\mathbf{p}||\mathbf{w}}^2 = \sum_{c=1}^C (p_c - w_c)^2 / w_c$. According to the authors, this error only disappears when $p_c = w_c$ or $\kappa^2 = 0$, which means either that all clients perform the same number of steps with the same local solver or that all local functions are identical (see Equation A.4).

Considering theorems 1 and 2, a straightforward solution emerges. One can simply replace w_c with p_c in the update rule (III.13) so that the term $\chi_{\mathbf{p}||\mathbf{w}}^2 \kappa^2$ becomes zero (i.e., the objective inconsistency vanishes). This modification leads to the FedNova algorithm.

Implementation

Algorithm 4 FedNova Algorithm

Functions of the framework

```

1  CLIENTOPT(*):
2     $\mathbf{w}_c^{(t,0)} = \mathbf{w}^t$ 
3     $\mathbf{G}_c^t =$  empty matrix of size  $\mathbb{R}^{d \times \tau_c}$ 
4    for epoch  $e = 0, 1, \dots, E_c$  do
5      for each batch  $\mathcal{B}$  of size  $B$  do
6         $\tilde{\mathbf{g}}_c^t(\mathcal{B}) = \text{OPTIMIZER}(F_c(\mathbf{w}^{(t,e)}))$ 
7         $\vec{\mathbf{g}}_c^t = \eta_c \tilde{\mathbf{g}}_c^t(\mathcal{B})$ 
8         $\mathbf{w}_c^{(t,e+1)} = \mathbf{w}_c^{(t,e)} - \vec{\mathbf{g}}_c^t$ 
9        add  $\vec{\mathbf{g}}_c^t$  to  $\mathbf{G}_c^t$ 
10     end for
11   end for
12    $\mathbf{d}_c^t = \frac{\mathbf{G}_c^t \mathbf{a}_c^t}{\|\mathbf{a}_c^t\|_1}$ 
13   return  $(\mathbf{d}_c^t, \|\mathbf{a}_c^t\|_1)$ 
14
15  AGGREGATION(*):
16   return  $\sum_{c \in S^t} w_c \cdot \eta_s \mathbf{d}_c^t$   $\triangleright w_c = p_c$ 
17
18  SERVEROPT(*):
19    $\tau_{\text{eff}}^t = \sum_{c \in S^t} p_c \|\mathbf{a}_c^t\|_1$ 
20   return  $\mathbf{w}^t - \tau_{\text{eff}}^t \Delta^t$ 

```

Effects of the parameters

The FedNova algorithm is characterized by its modularity, allowing users to choose the normalizing factor, denoted as α . In its most basic form, one could set $\mathbf{a}_c^t = \{1, \dots, 1\}$. This configuration effectively parallels the SGD solver, akin to the FedAvg approach. Alternatively, by defining $\mathbf{a}_c^t = \{(1 - \mu)^{\tau_c - 1}, (1 - \mu)^{\tau_c - 2}, \dots, (1 - \mu), 1\}$, one can employ proximal SGD as the local solver, much in the spirit of FedProx.

At the heart of FedNova's operation, especially in the SGD scenario, is the normalization of local updates based on the depth of the local computation. For instance, if a client undergoes 4 local steps (i.e., $\tau = 4$), the updates it contributes will be normalized by a factor of 4. According to the algorithm's authors, this normalization process nudges the global model towards a more optimal point in the solution space.

The mechanism is illustrated in Figure III.3.

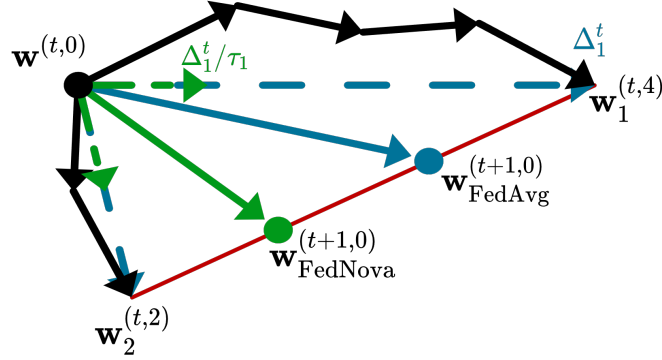


Figure III.3: Illustration of FedNova against FedAvg for 2 clients, $\tau_1 = 4, \tau_2 = 2$

The dotted blue arrows represent the client original updates, while the green ones depict the regularized updates. Solid green and blue arrows are the global model updates of FedNova and FedAvg, respectively. Based on the visual, it is clear that client 1 exerts a greater impact on FedAvg, leading to a global model biased towards client 1, in comparison to FedNova.

4 Incorporate server-side adaptivity

In the context of federated learning, momentum can address a significant challenge. Federated learning involves training models on data that is distributed across different clients with varying data distributions. This can lead to notable differences in the gradients computed by different clients, causing the model training to oscillate or converge slowly. By incorporating momentum into the algorithm, we can smooth out the gradients and reduce oscillations, leading to potentially faster convergence of the model.

Another benefit of using server-side momentum while keeping SGD as ClientOpt is that it does not affect the computational complexity at the client level, nor the communication cost per round. Additionally, adding momentum has a similar effect to selecting a larger set of clients, as the updated gradient in the next round still partially incorporates those of previously selected clients. [49]

Of particular interest are the Yogi and Adam optimization algorithms, which have been shown to be highly effective in improving the convergence rate and stability of model training in non-federated settings. Other optimization algorithms can be used as well, such as Adagrad or AdaDelta (the latter of which the pseudo-code can be

found in Algorithm 7). A comprehensive description of those optimizers is available in appendix ; see section 3.

Implementation

Remark. The notation is slightly abused for the sake of readability, by making an exception for $\Delta_t^2 = (\Delta^t)^2$. Also, the computation of v^t in SERVEROPT is performed only once, depending on the choice of the (sub-)algorithm.

Algorithm 5 Momentum server optimizers (Adagrad, Adam, Yogi)

Functions of the framework

```

1  AGGREGATION(*):
2       $\Delta^t = \sum_{c \in S^t} p_c \Delta_c^t$ 
3       $m^t = \beta_1 m^{t-1} + (1 - \beta_1) \Delta^t$ 
4      return  $(\Delta^t, m^t)$ 
5
6  SERVEROPT(*):
7       $v^t = v^{t-1} + \Delta_t^2$  ▷ Adagrad
8       $v^t = \beta_2 v^{t-1} + (1 - \beta_2) \Delta_t^2$  ▷ Adam
9       $v^t = v^{t-1} - (1 - \beta_2) \Delta_t^2 \text{sign}(v^{t-1} - \Delta_t^2)$  ▷ Yogi
10     return  $\mathbf{w}^t + \eta_s m^t \left( \sqrt{v^t} + \epsilon \right)^{-1}$ 

```

Effects of the parameters

The literature recommends default values of $\eta_s = 1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. When using AdaGrad, the values of β_1 and β_2 are set to zero $\beta_1 = \beta_2 = 0$ because Adagrad does not use momentum (see 3 for a more detailed theoretical explanation). The momentum optimizers are considered quite robust to hyperparameters changes.

One also need to choose the starting value of v and m . The original paper of those algorithms [23, 56] set them both to zero $v^0 = m^0 = 0$. On the other hand, Reddi et al. require in their analysis v^0 to be greater or equal than a *sensitivity* parameter usually set to 10^{-3} . Most of the implementations uses the bias-corrected moment estimates, while the federated implementation proposed by Reddi et al. does not.

5 Adding the privacy layer

5.1 Differential privacy

Differential privacy (DP) is a well-recognized method for establishing and quantifying the privacy constraints of sensitive data during learning processes. In essence, DP safeguards against adversaries who may attempt to infer if an individual's data was utilized in training a model, even when provided with arbitrary auxiliary information. The formal definition of DP is based on the concept of neighboring datasets, which refers to pairs of datasets $d, d' \in D$ that are considered neighbors if d' can be derived from d by adding or removing a single element.

Definition 3 (User differential privacy). Let $A : D \rightarrow R$ be a randomized algorithm with a domain D (training datasets) and output range R (trained models). The algorithm A is said to be (ε, δ) -user-level differentially private if, for all pairs of neighboring datasets $d, d' \in D$ differing by the data of one user, and for any subset of outputs $O \subseteq R$, the following inequality holds:

$$\Pr(A(d) \in O) \leq e^\varepsilon \cdot \Pr(A(d') \in O) + \delta \quad (\text{III.15})$$

This means that adding or removing a particular user's data from the training set barely affects the model's parameters distribution. Consequently, someone analyzing the finished model wouldn't be able to determine if data from a particular user contributed to its training, no matter the extra information they possess. Further, limiting the influence of any user is essential for privacy preservation and could even in some specific cases contribute to the system's stability. A common approach is to cap the L_2 norm of a client update vector Δ_c :

$$\pi(\Delta_c, M) \stackrel{\text{def}}{=} \Delta_c \cdot \min\left(1, \frac{M}{\|\Delta_c\|}\right) = \hat{\Delta}_c. \quad (\text{III.16})$$

For deep networks, the update of a client Δ_c may contain m layers with significantly different L_2 norms, making it more suitable to clip each layer individually, as suggested by McMahan et al. (2018). Given a per layer clipping parameter M_i , for each layer of Δ_c , we use $\pi(\Delta_{(c,i)}, M_i)$, with $M_i = \frac{M}{\sqrt{m}}$.

Upon establishing a bounded L_2 sensitivity, we can ensure DP by adding noise. The standard practice involves adding Gaussian noise to perturb the (stochastic) gradient in each training iteration. Further, as such clipping also effectively constrains the L_2 sensitivity of the aggregate with respect to the addition or removal of any user's data, introducing Gaussian noise to the aggregate is sufficiently adequate for obtaining a

central differential privacy guarantee for the update [1]. Subsequently, the privacy expenditure of the global model can be determined based on the privacy budget using RDP Accountant from Mironov, Wang, Balle, and Kasiviswanathan (2017, 2018).

Selecting an appropriate clipping threshold is crucial for the utility of the private training mechanism. If the threshold is set too low, high bias may arise from neglecting information contained in the gradient’s magnitude. On the other hand, setting it too high necessitates the addition of more noise, as the Gaussian noise required for a given privacy level must be proportional to the norm bound (the L_2 sensitivity), which could ultimately compromise model utility.

The challenge lies in the fact that each hyperparameter change, such as the number of selected clients, batch size, client and server learning rates, weighting scheme, and so on, has a direct or indirect impact on the strength of the measures to be applied to preserve a certain level of privacy when there is no prior information available about the system. In order to identify the optimal clipping norm for all studied problems, it would be necessary to search over a very large range of clip values.

DP-FedAvg with adaptive quantile clipping

A method proposed by Andrew et al. (2022), which offers an approach for adaptively and privately adjusting the clipping threshold to track a specific quantile of the update norm distribution during the training process. The authors claim that this method consumes a minimal amount of privacy budget and is compatible with other FL technologies, such as compression and secure aggregation.

The primary advantage of this approach, compared to earlier methods, is the adaptive selection of the clipping parameter during training without the need for prior knowledge about the upper bound of the gradient norm. This obviates the necessity for hyperparameter search concerning M . The objective is to target the quantile $\gamma \in [0, 1]$, employing a geometric update rule for adjusting the clipping norm:

$$M^{t+1} = M^t \cdot \exp\left(-\eta_M(\tilde{b}^t - \gamma)\right), \quad (\text{III.17})$$

where $\tilde{b}^t$ denotes the proportion of clients with clipped updates. To prevent the disclosure of average update amplitude, Gaussian noise with a standard deviation of $\sigma_b = |S^t|/20$ is introduced.

The noise level incorporated into user updates is proportional to M and scaled by a noise multiplier, z . In conventional approaches [36], the multiplier is typically set to one. In this case, the variable is updated as follows:

$$z_{\Delta} = \left(z^{-2} - (2\sigma_b)^{-2} \right)^{-1/2}, \quad (\text{III.18})$$

resulting in a marginally higher noise multiplier to account for adaptiveness.

Implementation

The authors recommend using the median as the target, $\gamma = 0.5$, an initial value of $M = 0.1$, and a learning rate of $\eta_M = 0.2$ for all tasks. The implementation proceeds as follows:

Algorithm 6 Fed-DP Algorithm

Functions of the framework

```

1  CLIENTOPT(*):
2     $\mathbf{w}_c^{(t,0)} = \mathbf{w}^t$ 
3    for epoch  $e = 0, 1, \dots, E_c$  do
4      for each batch  $\mathcal{B}$  of size  $B$  do
5         $\tilde{\mathbf{g}}_c^t(\mathcal{B}) = \text{OPTIMIZER}(F_c)$ 
6         $\mathbf{w}_c^{(t,e+1)} = \mathbf{w}_c^{(t,e)} - \eta_c \tilde{\mathbf{g}}_c^t(\mathcal{B})$ 
7      end for
8    end for
9     $\Delta_c^t = \mathbf{w}_c^{(t,E_c)} - \mathbf{w}^t$ 
10    $\hat{\Delta}_c^t = \pi(\Delta_c^t, M^t)$ 
11    $b_c^t = \mathbb{I}_{\|\Delta_c^t\|_2 \leq M^t}$ 
12   return  $(\hat{\Delta}_c^t, b_c^t)$ 
13
14 AGGREGATION(*):
15    $\sigma_{\Delta} = z_{\Delta} M^t$ 
16    $\tilde{\Delta}^t = \sum_{c \in S^t} p_c \hat{\Delta}_c^t + \mathcal{N}(0, I \sigma_{\Delta}^2)$ 
17    $\hat{b}^t = \sum_{c \in S^t} b_c^t$ 
18    $\tilde{b}^t = \frac{\mathcal{N}(\hat{b}^t, \sigma_b^2)}{|S^t|}$ 
19   return  $(\tilde{\Delta}^t, \tilde{b}^t)$ 
20
21 SERVEROPT(*):
22    $M^{t+1} = M^t \cdot \exp\left(-\eta_M(\tilde{b}^t - \gamma)\right)$ 
23   return  $\mathbf{w}^t + \eta_s \tilde{\Delta}^t$ 
```

$$\triangleright p_c = \frac{1}{|S^t|}$$

Chapter IV

Implementation & Research

1 Datasets

The performances of the proposed federated learning algorithms were evaluated using some datasets and tasks used by many researchers, as they are some of the ubiquitously known datasets. With the exception of CIFAR10, all are non-i.i.d. with respect to user partitioning : the data is partitioned according to the “user who generated the data”, for example the writer of the FEMNIST characters or speaking role in Shakespeare’s plays.

1.1 CIFAR-10

The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) is a widely used benchmark in the field of machine learning, particularly for image classification tasks. It comprises 60,000 color images, each of 32x32 pixels in size, distributed across ten distinct classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). Each class is represented by 6,000 images, with a total of 50,000 images allocated for training and 10,000 images reserved for testing purposes. Figure IV.1 provides a visual representation of sample images from the CIFAR-10 dataset, prior to any preprocessing.

In order to match the FedAvg paper, the following preprocessing steps are performed on the dataset:

- Training images: random crop to shape (24, 24, 3) and random horizontal flip.
- Test images: central crop to shape (24, 24, 3).
- Both training and testing images: normalize pixel values using mean and standard deviation.

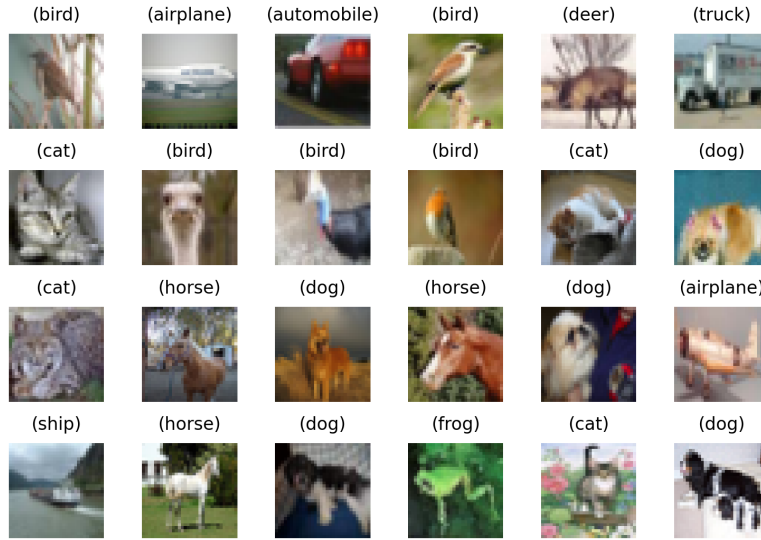


Figure IV.1: CIFAR10 random samples, labels on top

However, the natural structure of the dataset does not exhibit the properties of a non-iid data distribution, which is commonly observed in real-world applications. To overcome this limitation and more accurately simulate realistic data distribution settings, we employed the LDA-based sampling technique proposed by Hsu, Qi, and Brown (2019) and applied in “Adaptive Federated Optimization” to create non-iid distributions for the CIFAR-10 dataset.

This Latent Dirichlet Allocation (LDA) implementation leverages a Dirichlet distribution to adjust the class distribution. The behavior is controlled by a parameter, α_1 . Further, we introduced an additional parameter, α_2 , to manipulate the distribution of samples per client unevenly, also using a Dirichlet distribution for selection. As α_1 increases, the class distribution becomes more balanced across clients, while a higher α_2 results in a more equitable distribution of samples.

Artificially injecting non-i.i.d. characteristics into CIFAR10 in a regulated way offers an opportunity to study the impact of non-iidness on the learning mechanism. In our experiments, we decided to push the heterogeneity to quite a high level, varying α_1 and α_2 up to 0.5. Figures IV.7a and IV.7b illustrate the sampling mechanism.

1.2 FEMNIST

The Federated Extended MNIST (FEMNIST) dataset comes from nist.gov, and is a challenging dataset for federated learning algorithms due to the large number of classes and the variability in the number of samples contributed by each writer. The dataset consists of 814,277 grayscale images (28x28) of handwritten characters, including digits, lowercase and uppercase letters, spread across 62 classes. The dataset comprises images sourced from 3,597 writers.

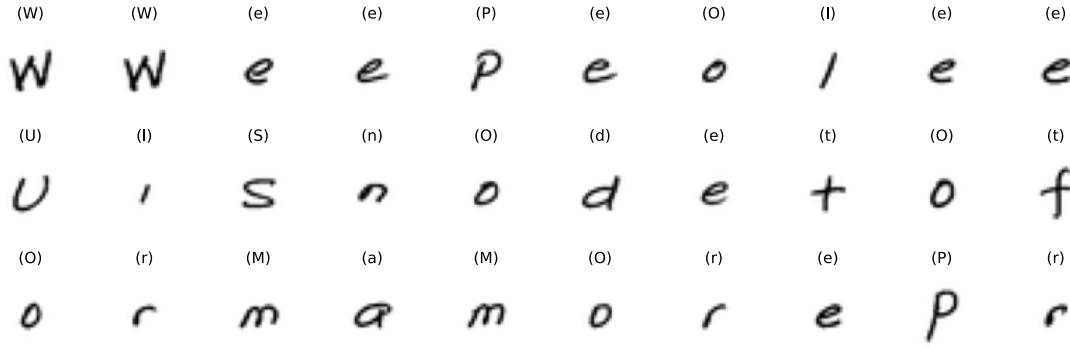


Figure IV.2: Samples of one writer, labels on top

Notably, each writer has contributed a varied number of samples, leading to an uneven distribution. This variability, or disparity in the number of samples from each writer, is clearly depicted in Figure IV.3b. Furthermore, the classes within the dataset aren't uniformly distributed and certain digits or letters are more prevalent than others, introducing an additional layer of non-IID characteristics to the dataset.

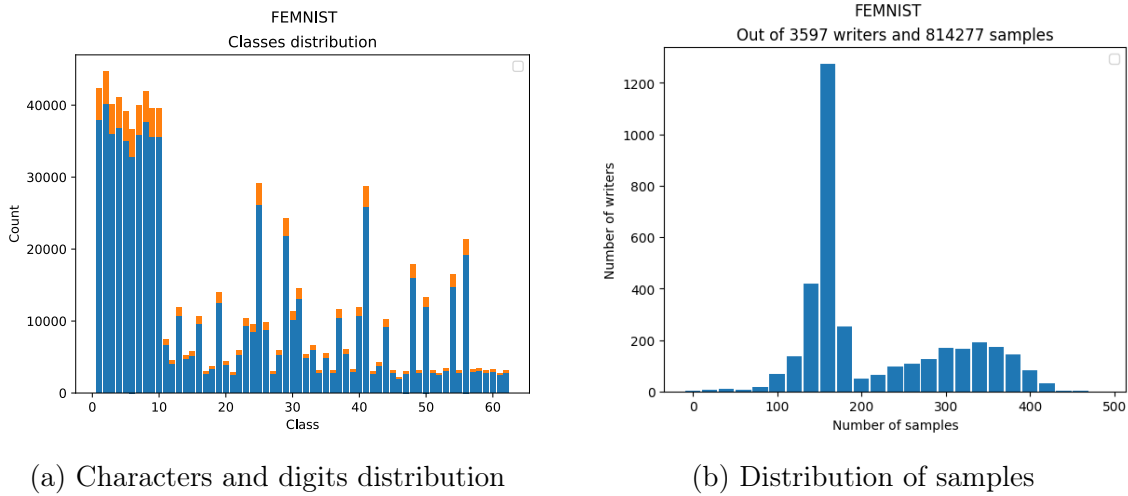


Figure IV.3: FEMNIST Dataset

1.3 Shakespeare

The dataset consists of text data from the complete works of William Shakespeare, including his plays and sonnets, providing a diverse range of linguistic styles and speaking idiosyncrasies. The dataset is made of by 660 speaking roles, considered as individual clients, with a really high disparity in the amount of data. There is 3,678,451 samples in the training set and 356,921 samples in the test set. Inherent to its structure, the dataset also displays non-IID characteristics, primarily due to correlated labels. Figure IV.4a shows the distribution of the number of training samples per client. The lowest value is 721, while the highest is 60,212.

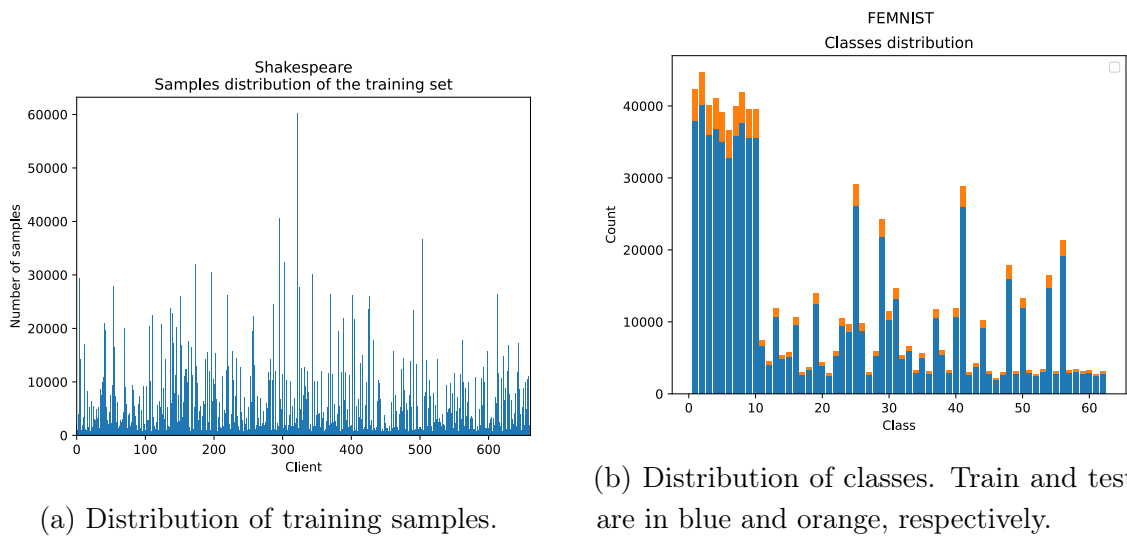


Figure IV.4: Shakespeare dataset

The task with this dataset is next-character prediction. Given a rolling sequence of 80 characters, the goal is to predict the character that follows. For example, those 6 first inputs from the speaking role *Bertram*, from the book *All's Well That Ends Well* have a, j, e, s, t, y as prediction:

And I in going, madam, weep o'er my father's death anew; but I must attend his M
nd I in going, madam, weep o'er my father's death anew; but I must attend his Ma
d I in going, madam, weep o'er my father's death anew; but I must attend his Maj
I in going, madam, weep o'er my father's death anew; but I must attend his Maje
I in going, madam, weep o'er my father's death anew; but I must attend his Majes
in going, madam, weep o'er my father's death anew; but I must attend his Majest

Each character of the sentence is encoded as the index in a table mapping the character to an integer. The predictions on the other hand are one-hot encoded.

2 Experiments

2.1 Setup

The experimental environment consisted of an NVIDIA RTX 3080 GPU, Intel 11700KF@4.50GHz CPU, and 96GB of RAM at 3000MHz, using Python 3.10.9 and Pytorch 2.0.1+cu117. The distributed learning aspect was handled by the open-source framework Flower [3]. For all experiments, the same random seed was used.

Evaluation

To evaluate the performance of the federated learning algorithms, we implemented the federated evaluation method, as presented in subsection 4.3. The evaluation was done every round on another sample of client than for training. Moreover, once every 5 rounds, we (unrealistically) evaluated the performance on 100% of the clients test data in order to see the un-noisy global performance.

	CIFAR10	FEMNIST	SHAKESPEARE
Accuracy	0.50	0.70	0.45
Train	10/200 (5%)	48/3597 (1.33%)	10/660 (1.52%)
Test	20/200 (10%)	48/3597 (1.33%)	30/660 (4.55%)

Table IV.1: Satisfactory accuracies considered in our experiments and amount of training and testing clients selected per round.

Given the specificity of this research towards classification tasks, the cross-entropy loss function (with sum reduction)¹ is employed. The raw performance of a model is subsequently quantified through the loss as well as the top-1 accuracy.

Another central challenge arises from the fact that all devices, including those with insufficient data or suboptimal systems, should still yield robust prediction performances. In an ideal scenario, the global model would demonstrate uniformly strong performance across all devices [25, 22]. This notion called the (individual) *fairness* and a model $\mathbf{w}$ is said to be more fair than $\mathbf{w}'$ if the variance in terms of performance measures (ie. accuracy) is lower across clients. We will compare the fairness between algorithms.

¹Refer to Pytorch’s documentation of `torch.nn.CrossEntropyLoss` for a detailed explanation.

Stragglers and dropping clients

To simulate system heterogeneity and account for the possibility of clients being unable to complete the requested number of epochs within a given time window (i.e., $E_c \leq E$), we intentionally introduced stragglers among the clients. At each round, a certain number of clients are designated as stragglers, performing between 1 and the requested number of epochs uniformly at random, i.e., $E_c \sim \mathcal{U}(1, E)$. The number of stragglers varies between rounds but gravitates around a fixed mean. In the end, this approach ensures that we achieve the desired percentage of dropouts and stragglers. We also added the possibility for some clients to drop out and send nothing back to the server, using the same mechanism.

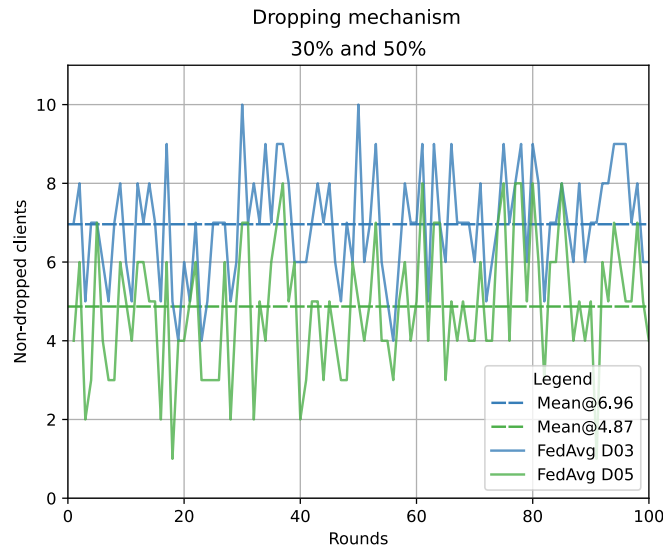


Figure IV.5: Amount of not-dropped clients per round. 10 clients selected and 100 rounds, with 30% and 50% (in expectation) of client dropped. Also valid for the stragglers selection mechanism.

This study defines six distinct scenarios to explore the multifaceted dimensions of system heterogeneity:

- **Basic:** An idealized situation where all clients complete the requested number of epochs without disconnection.
- **S05** and **S09:** Scenarios with 50% and 90% of stragglers, respectively.
- **D03** and **D05:** Scenarios with 30% and 50% of disconnections, respectively.
- **S09D05:** A more harsh situation with half the clients (50%) not responding, and almost all (90%) not fulfilling expectations.

Clients learning rates and optimizers

As we implemented the client as stateless, the instance of a particular client is recreated every time the client is selected, and both their optimizer and scheduler states are reset. We opted for SGD as the optimizer and implemented a simple REDUCELRONPLATEAU² scheduler, which decreases the learning rate after 2 epochs of plateau in the loss by a factor of 0.1.

Selecting and finding the optimal learning rate is difficult and varies between datasets and even algorithms.³ For instance, we used client learning rates as depicted in Table IV.2. For the sake of simplicity, we maintained the same values for all the algorithms within the same experiment. The goal is not to train the best possible model, and we lack the computational capability to extensively search for the best learning rate for each algorithm and dataset. We used a small search grid of $\{0.01, 0.03, 0.1, 0.3\}$ for the client learning rate in the basic setting and found the values in Table IV.2 to yield the best accuracies.

	CIFAR10	FEMNIST	SHAKESPEARE
LR	0.01/0.003	0.03	0.1
Batch	16	16	8
Epoch	10	5	5

Table IV.2: Client learning rates η_c , requested epochs E and batch-size B .

Server learning rate

On the other hand, the server learning rate is stateful. We experimented with different techniques and opted for the DEMON strategy by Chen et al. (2021) [9], with a starting value of $\eta_s = 0.95$. We chose this strategy as it is “computationally cheap, understandable, easy to implement, and offers better performances”. The function is shown in dashed purple in Figure IV.6 and described in Equation IV.1 with t the current round, T the total amount of rounds, and η_s the initial server learning rate :

$$\eta_s^t = \eta_s \cdot \frac{(1 - \frac{t}{T})}{(1 - \eta_s) + \eta_s(1 - \frac{t}{T})} . \quad (\text{IV.1})$$

²See the Pytorch documentation of `torch.optim.lr_scheduler.ReduceLRonPlateau`

³We refer the reader to Table 8 of “Adaptive Federated Optimization” [40], which summarizes the best-found learning rates by the authors for various datasets.

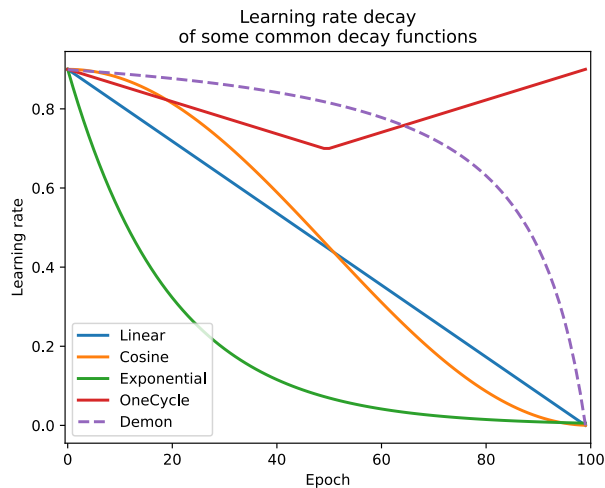


Figure IV.6: Illustration of common learning rates decay strategies, starting at 0.9. The exponential function has a factor of 0.95.

The rationale behind using a decaying learning rate is that, in the early stages of optimization, a relatively large learning rate can facilitate faster convergence towards a suitable region in the parameter space. As the optimization progresses, the algorithm approaches the optimal solution, and a smaller learning rate is beneficial to avoid overshooting the optimal point. Further, it has been shown that, under certain conditions, using a decaying learning-rate strategy can guarantee convergence to a stationary point or an approximate local minimum in noisy or non-convex problems (Li et al. (2019)), which are often encountered in federated learning.

Models

For the CIFAR10 dataset, the ResNet-18 model was initially used, but we later transitioned to the TwoCNN model with 32 hidden layers due to its ease of training and superior performance. When working with the FEMNIST dataset, the TwoCNN model was also the choice of implementation. For the Shakespeare dataset, the NextCharLSTM model was employed, handling an input of 80 characters. These specific models were selected as they have been utilized in many previous works and therefore offer a basis for comparison and alignment with established practices. All the architectures are available in appendix ; see section 2.

2.2 Statistical heterogeneity analysis

To begin, we find it compelling to analyze the effect of heterogeneity at the data level in a controlled manner. More specifically, we will start by training the CIFAR-10 dataset while varying the sampling mechanism to observe the impact of imbalanced data and label distribution skew (see subsection 4.1).

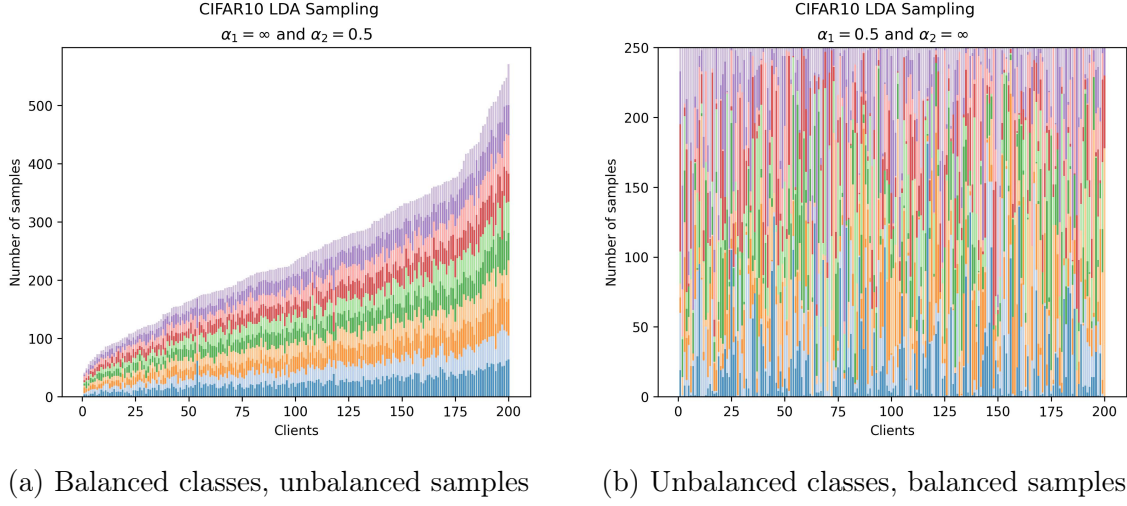


Figure IV.7: Semi-balanced training sets, comparison of the effect of α_1 and α_2

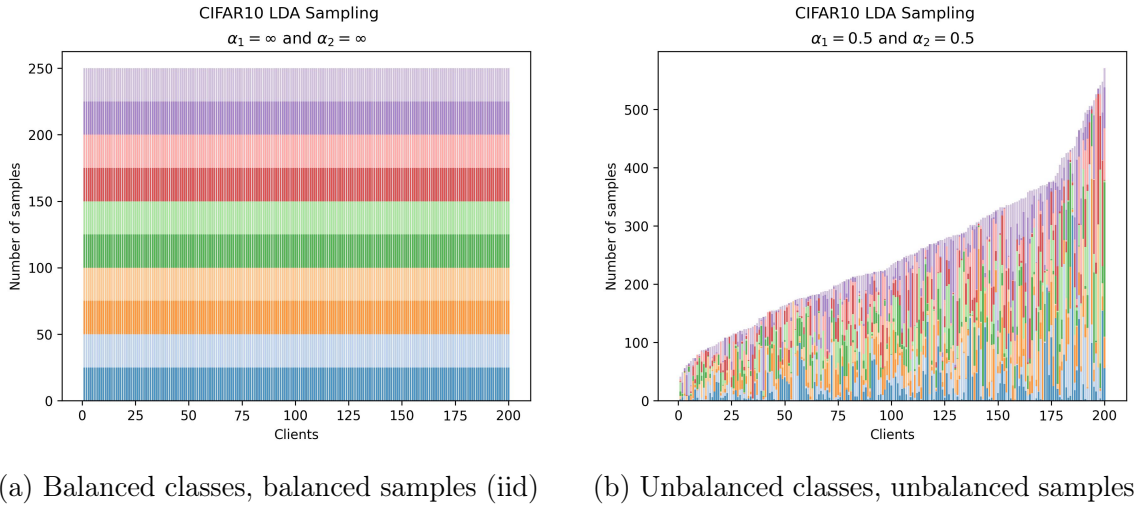


Figure IV.8: Comparison of fully balanced and fully unbalanced training sets.

In Figure IV.9, we can observe that samplings with a balanced class distribution outperform the balanced distribution of data quantities, as one might expect. A surprising fact however is that the iid (inf-inf) distribution is lower than the semi-balanced distribution (inf-0.5).

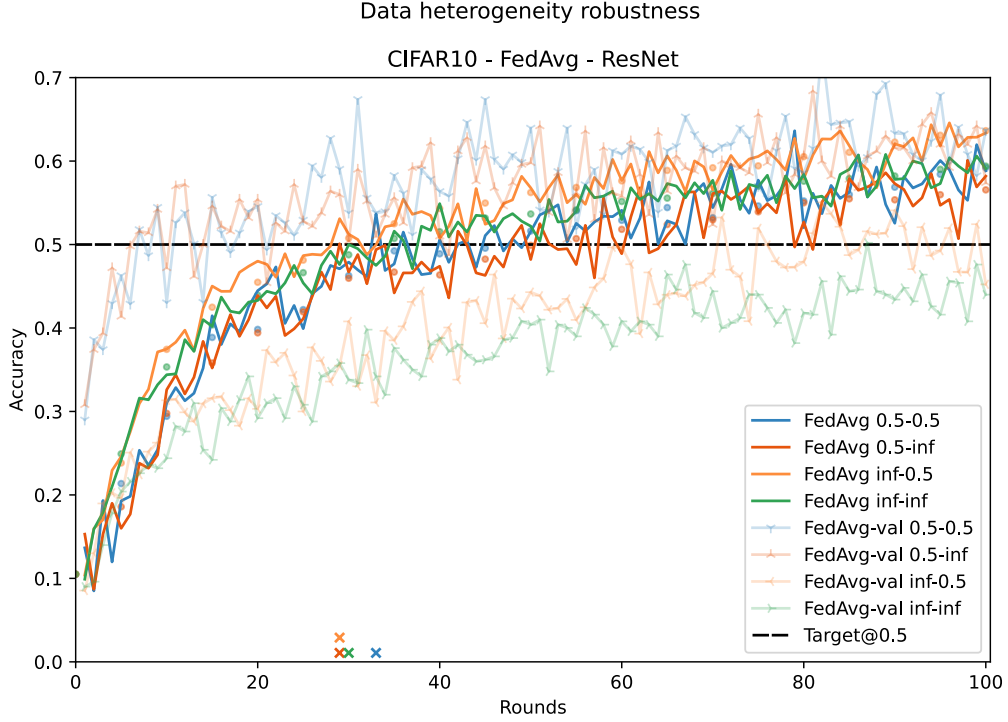


Figure IV.9: Test and validation accuracies of FedAvg for the CIFAR10 dataset with 4 different sampling strategies and heterogeneity levels. The crosses at the bottom helps visualise when the algorithms goes over the threshold. The colored dots are the accuracies on the full set of clients. ResNet model.

Displayed in transparency is the accuracy on each client’s validation set after local training. We notice that the trend reverses and does so much more sharply. Our hypothesis is that this might be due to overfitting of local models when classes are unbalanced, since some classes are consequently not trained locally. This results in higher validation accuracies (validation being based on the training set) but lower test accuracies.

In Figure IV.9, the IID dataset (represented by the green curve) and the fully non-IID dataset (shown by the blue curve) seem quite alike at first glance. However, when we focus on the individual accuracies of each client, a clear distinction emerges. The IID distribution, as seen in Figure IV.10a, is notably more balanced and gravitates around the average. In contrast, with a heterogeneous dataset, there’s a wide spread of client accuracies, as depicted in Figure IV.10b. For more visuals and a fairness

comparison, refer to the appendix at Figure 3.2 and Table B.4.

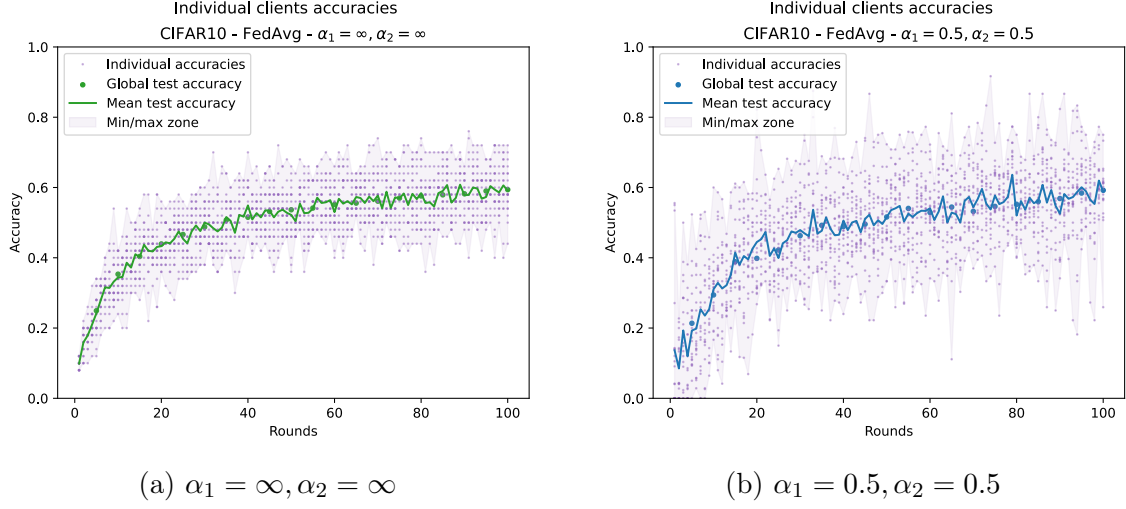


Figure IV.10: Test accuracies of selected clients for training. Note that clients are not the same between rounds. The accuracy measure is done after the aggregation stage.

Furthermore, non-IID sampling of the classes presents other disadvantages that are not evidently seen in metrics such as accuracy or loss. To illustrate this, let's observe the confusion matrix for the others two scenarios. Although the two models are only separated by a few percentage points in accuracy, we can see that the cat predictions are biased and unusable in the unbalanced classes one (Figure IV.11a), while this issue is not happening in the unbalanced samples (Figure IV.11b) ; further reinforcing our observations.

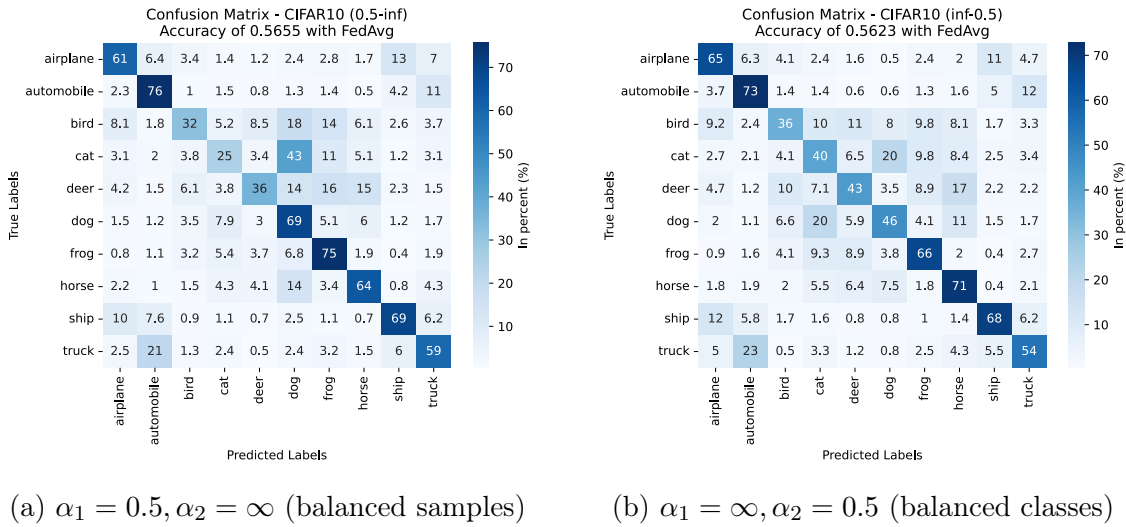


Figure IV.11: Confusion matrix for predictions made with the best trained FedAvg model on the CIFAR10 full test dataset. Numbers in percentages, normalized by row. 50 rounds.

Lastly, Figure IV.12 presents the full-test accuracies for each algorithm and sampling scenario. First, we notice that our previous observations hold true and that the class distribution is the dominant factor affecting the final accuracy. We now understand that the class distribution has a far greater impact than the number of data points, up to a certain limit (a client with too few data will accomplish nothing).

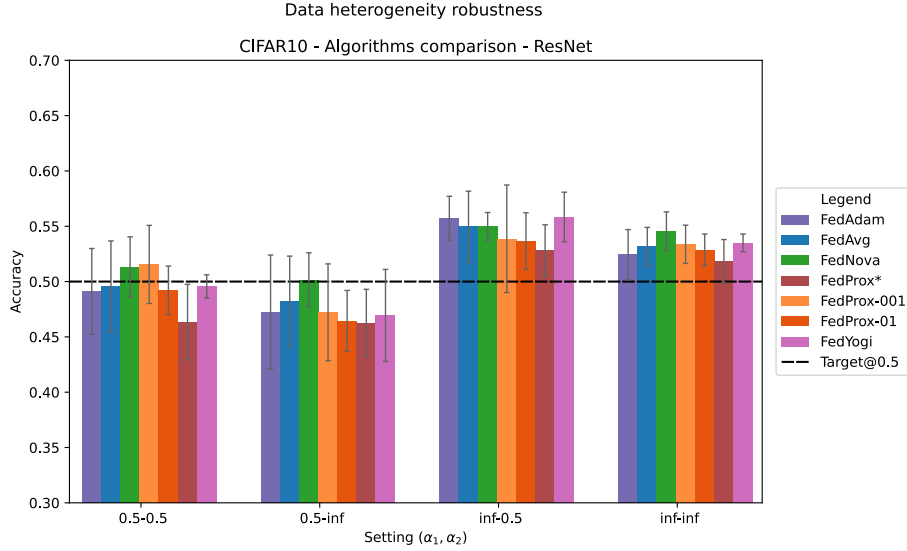


Figure IV.12: Barplot comparing the accuracy on the full test set, 100 rounds. Errors are the absolute difference between the full test and (biased) distributed evaluation.

Another thing that stands out right away is that the algorithms don't seem to differ much between them, no matter how varied the data is. This is interesting, especially when thinking about what the authors had originally claimed about their algorithms. Now, let's see how many rounds each algorithm takes to hit our target accuracy of 0.50 on the test dataset:

	0.5-0.5	0.5-inf	inf-0.5	inf-inf
FedAdam	32	33	22	29
FedAvg	32	28	28	29
FedNova	32	28	24	26
FedProx($\mu = 0.01$)	32	41	26	26
FedProx($\mu = 0.1$)	32	50	27	33
FedProx*($\mu = \text{adaptive}$)	50	50	30	35
FedYogi	35	32	27	29
Best relative to FedAvg	0	0	6	3

Table IV.3: Comparison of the rounds needed to achieve the fixed threshold on the different CIFAR10 settings and algorithms. Resnet model.

This comparison illustrates that, while the overall differences are minor, certain techniques may lead to improvements in specific scenarios. Nevertheless, by examining the progression curves for each dataset (see subsection 3.3), we cannot conclude that any particular algorithm is significantly faster or more robust than the others. In Figure B.3, we notice that when evaluating across all clients, every algorithm surpasses the threshold by round 30. The only algorithm that could be distinguished from the rest is FedProx* with adaptive μ , which underperforms.

However, this may be attributed to an insufficient number of training rounds for the ResNet model⁴. With this in mind, we considered training with the TwoCNN model which possesses fewer parameters, to see if using a simpler model would make the algorithms behave differently. The results were indeed remarkable; as depicted in Figure IV.13, the TwoCNN model not only surpassed the necessary rounds to attain an accuracy of 0.5 but effortlessly reached a 0.65 accuracy mark. On the other hand, we notice that all algorithms of the same model continue to display similar performance levels. A barplot comparing the two models best accuracies is available in appendix, see Figure B.7.

⁴For instance, Reddi et al. (2020) [40] trained 4000 rounds on CIFAR10.

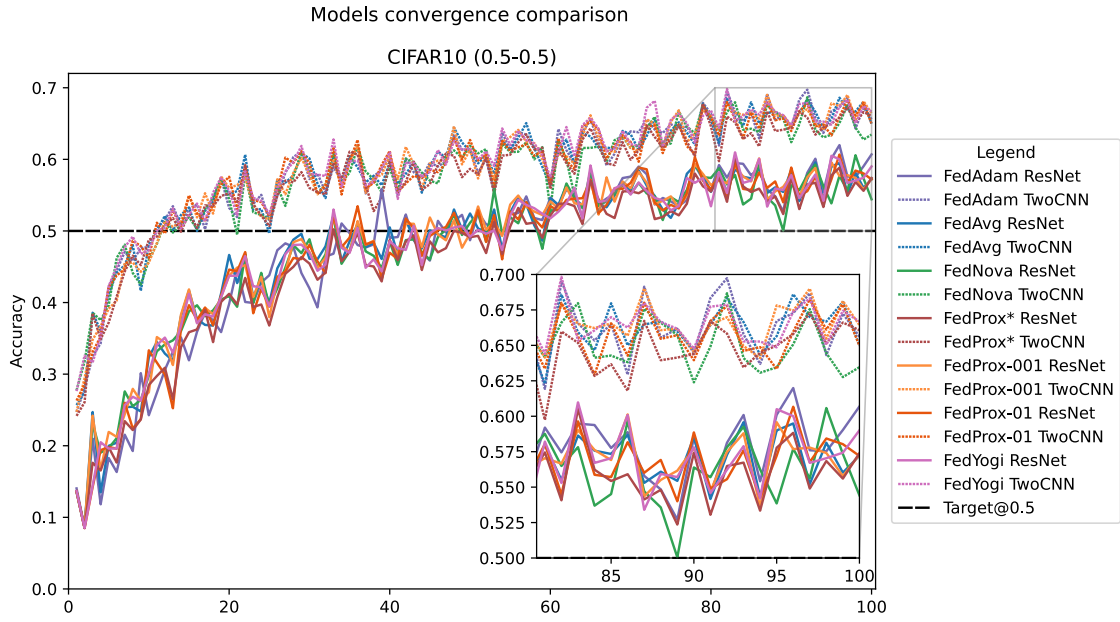


Figure IV.13: Comparison of the distributed test accuracy for the ResNet and TwoCNN models on the CIFAR10 (0.5-0.5) dataset.

2.3 Robustness to system heterogeneity

In this section, we turn our attention to client-level heterogeneity, focusing on the robustness of algorithms to variations that occur in local training. Since the FEM-NIST and Shakespeare datasets are naturally partitioned in a non-iid manner across clients, it is an appropriate starting point to observe the impact of inconsistencies or weaknesses at the client level for these two datasets.

Remark. Note that for the following figures, the FedAvg algorithm is configured not to drop stragglers, in contrast to what the theory prescribes. We made this decision because our goal is to evaluate the aggregation mechanism itself, rather than the design choice of whether or not to drop stragglers.

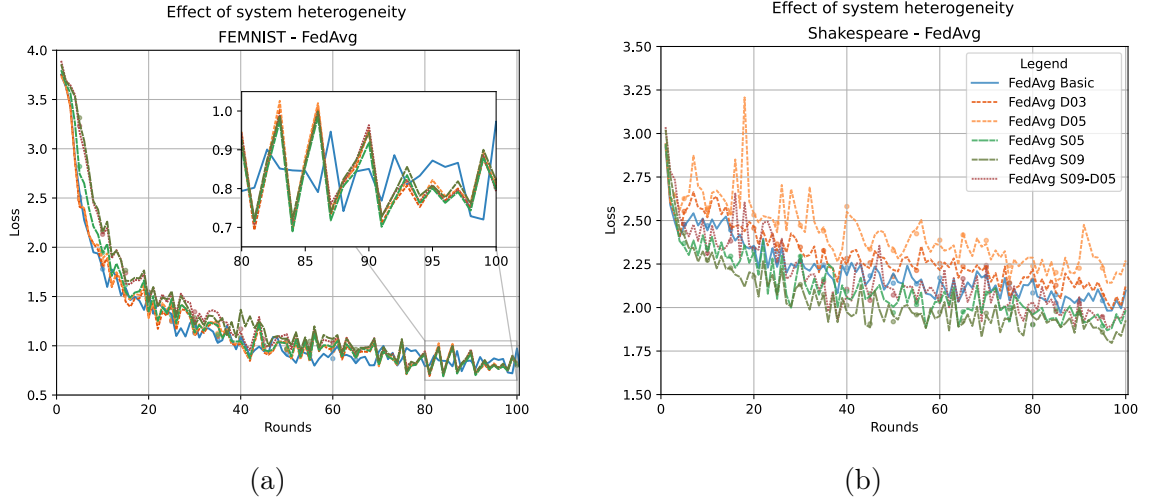


Figure IV.14: Convergence of FedAvg for FEMNIST and Shakespeare datasets across scenarios. The legend is common to both figures, see (b).

We can identify two distinct effects between the datasets. In the case of the FEMNIST dataset, as illustrated in Figure IV.14a, the influence of client-level heterogeneity is marginal, leading only to a slightly elevated loss in the first rounds when stragglers are part of the equation. This effect is most noticeable in the S09 scenario, though it remains relatively minor.

Contrastingly, the Shakespeare dataset exhibits the opposite behavior. Here, the presence of stragglers results in a reduction in loss, even when the asked amount of local work is as low as just 5 epochs. Additionally, we have empirically found that the Shakespeare dataset has a tendency to diverge when subjected to a certain degree of local work. This finding aligns with the observations made by Caldas et al. (2018) [8].

FEMNIST

Before delving into the system robustness of the algorithms, let's first observe how they behave in the ideal scenario. As shown in Figure IV.15, on the FEMNIST dataset all the algorithms reach the desired threshold after just 30 to 40 rounds and manage to progress up to 0.8 accuracy if the training is continued for 150 rounds. Only FedNova appears to slightly stand out by being on the higher side of the curves, but does not seem able to keep its lead in the long run.

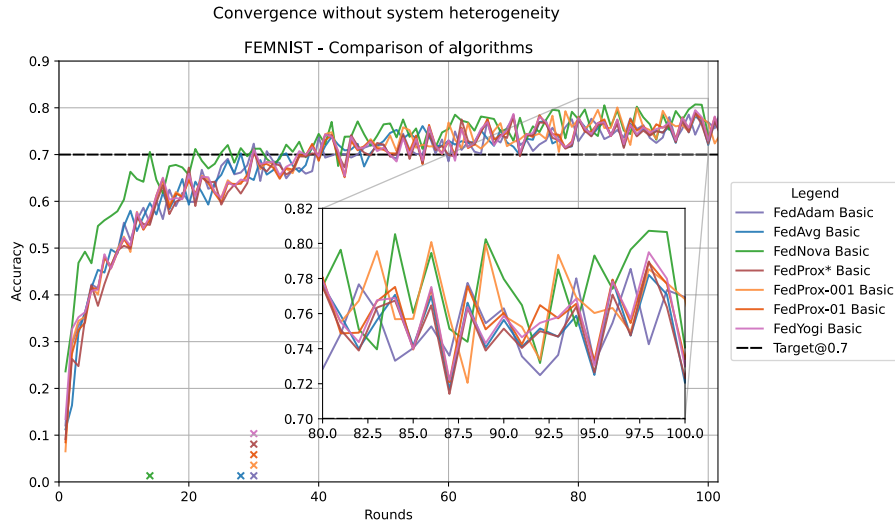


Figure IV.15: Accuracy convergence of algorithms on FEMNIST. Basic scenario.

When we take a closer look at the harder, S09-D05 scenario, shown in Figure IV.16, we can easily see how the algorithms behave under tougher conditions. Right away, we notice that all the algorithms do really well. This time, as FedNova was a bit ahead in Figure IV.15, we decided to try combining it with FedAdam. As we can see, this was a really good idea and it reaches the threshold far sooner than the others. Further, it is even able to keep its advantage until the end.

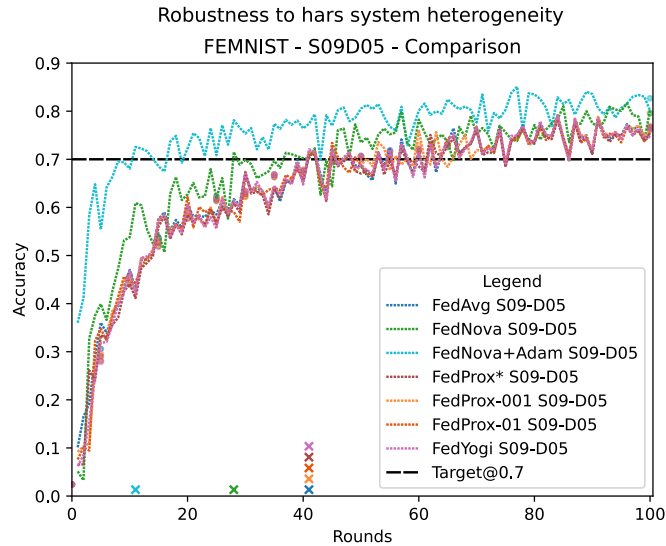


Figure IV.16: Accuracy convergence of algorithms on FEMNIST. Scenario with 90% stragglers and 50% of devices dropping out.

In general, all the algorithms take more time to get good results in this scenario. They need around 40 to 50 rounds, which is twice as long as in the basic case. This

pattern is pretty much the same in the other settings. FedNova tends to do a bit better at first, but the other algorithms catch up eventually. FedNova+Adam is really impressive, no matter the scenario. Detailed visual representations for D03, D05, S05, and S09 can be found in the appendix under subsection 3.5.

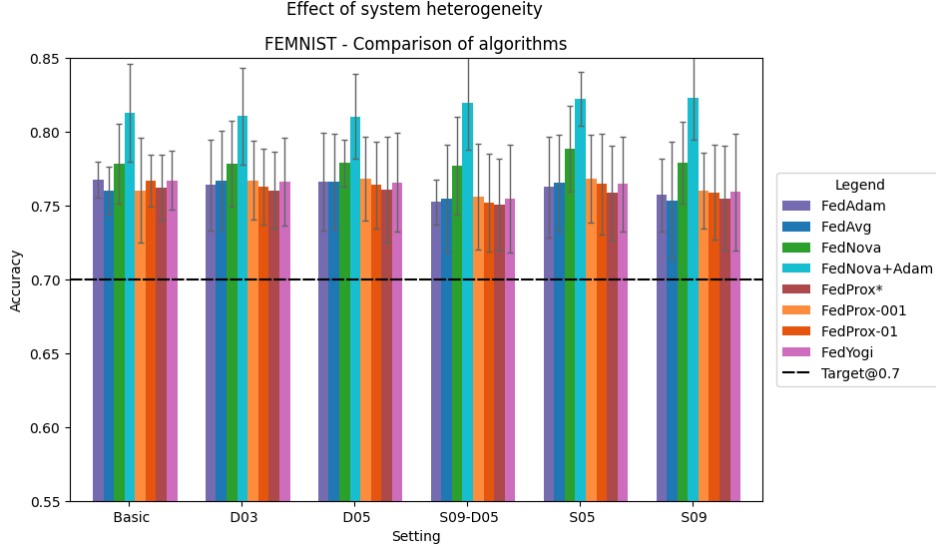


Figure IV.17: Barplot of the best accuracy in 100 rounds on the full test set.

When evaluating the peak accuracy achieved on the entire test set after 100 rounds, it becomes evident that system heterogeneity exerts minimal to no influence on the outcomes for the FEMNIST dataset. The performances of the algorithms are strikingly similar with the exception of FedNova, and especially FedNova+Adam which stand out, consistently outpacing the others in every setting. Interestingly, the presence of stragglers seems to slightly further accentuate the lead of FedNova+Adam. This enhanced performance might be attributed to the added momentum, showcasing the synergy between FedNova and Adam in handling straggler-induced challenges.

Next, let's dive into fairness. We assess fairness by checking the accuracy on each client's test data from three angles: the difference between the lowest and highest accuracy (gap), the variance, and the distribution. The standout point from Table IV.4 is the noticeable high values. These high figures are indicative of the variability among different clients' accuracies, which is undesirable.

	Basic	S05	D05
FedAdam	0.8889 / 0.0202	0.9444 / 0.0221	0.9444 / 0.0225
FedAvg	0.9412 / 0.0204	0.9444 / 0.0220	0.9091 / 0.0231
FedNova	0.8571 / 0.0143	0.8000 / 0.0151	0.8889 / 0.0181
FedNova+Adam	0.8000 / 0.0141	0.9286 / 0.0121	0.9286 / 0.0141
FedProx($\mu=0.01$)	0.9412 / 0.0203	0.9444 / 0.0219	0.9091 / 0.0221
FedProx($\mu=0.1$)	0.9412 / 0.0202	0.9444 / 0.0224	0.9091 / 0.0228
FedProx*($\mu=\text{adaptive}$)	0.9091 / 0.0207	0.9444 / 0.0228	0.9091 / 0.0231
FedYogi	0.8889 / 0.0200	0.9444 / 0.0226	0.9444 / 0.0231

Table IV.4: Comparison of two fairness metrics on the FEMNIST dataset, shown as **gap** / **variance**. Lower is better. Results for D03, S09, S09D05 are similar.

This table offers a compelling view into the performance of the algorithms. On one hand, it's clear that FedNova, whether combined with Adam or not, doesn't just excel in accuracy but also stands out in terms of fairness. When we shift our focus to variance, FedNova paired with Adam takes the lead. However, it's intriguing to note that introducing momentum via Adam seems to slightly widen the accuracy gap in scenarios with some degree of client heterogeneity. This suggests that while momentum aids in convergence, it might introduce challenges in ensuring consistent accuracy across diverse client datasets.

In all the scenarios and across all algorithms, the smallest difference in accuracy observed is 0.80. To be more specific, out of the 3,597 clients, the worst one only manages to get an accuracy of 0.20 when predicting its 5 samples. The obtained predictions are [w, a, p, v, E], while the true labels are [n, a, P, V, I]. However, by considering the top-3 predictions the accuracy increases to 0.8, with only the first sample not recognized as n.

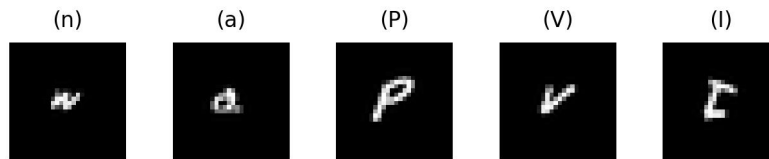


Figure IV.18: The 5 test samples of the client with the lowest accuracy

A comparison of the FedNova test accuracies distribution against the baseline algorithm is depicted in Figure IV.19. One can observe that FedNova’s histogram is shifted to the right relative to FedAvg, and FedNova+Adam is even more right-skewed. Furthermore, in average across all scenarios, only 71.10% of clients achieve an accuracy of at least 0.7 (our threshold) with FedAvg, in contrast to 79.49% for FedNova and 86.83% for FedNova+Adam. Even for FedNova, an 8% improvement is undeniably significant. Others algorithms, such as FedProx, does not show that much improvements as compared to FedAvg (see subsection 3.6 in appendix).

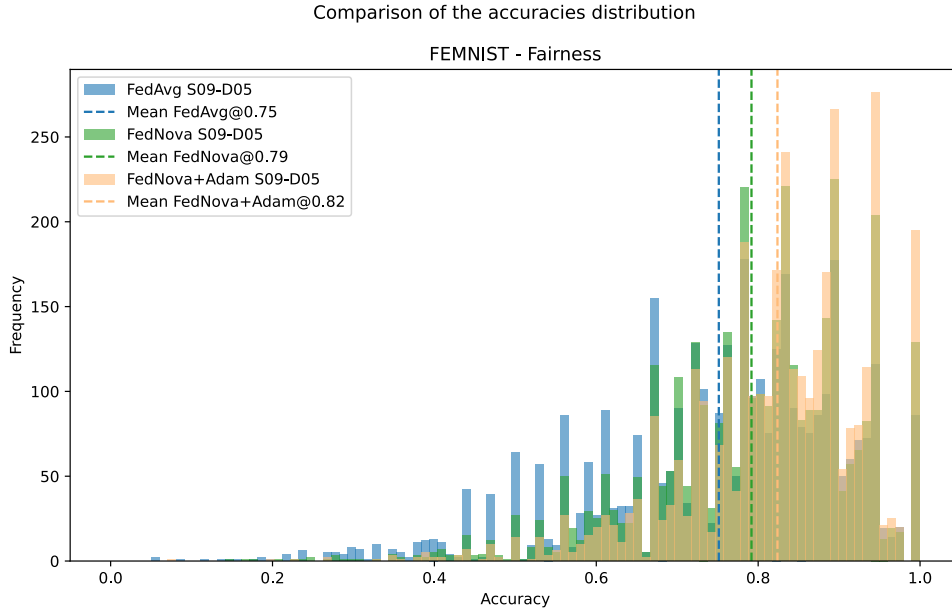


Figure IV.19: Fairness histogram comparing FedAvg to FedNova and FedNova+Adam in the S09D05 scenario of FEMNIST. 3597 clients. Mean accuracies are shown with vertical lines. The color of FedNova+Adam is different for the sake of visibility.

ShakeSpeare

FedNova has shown impressive results with the FEMNIST dataset. But how does it fare with a text dataset? In light of our earlier findings (Figure IV.14b), we expect that scenarios featuring stragglers could yield outcomes that are on par with, or potentially superior to the basic one.

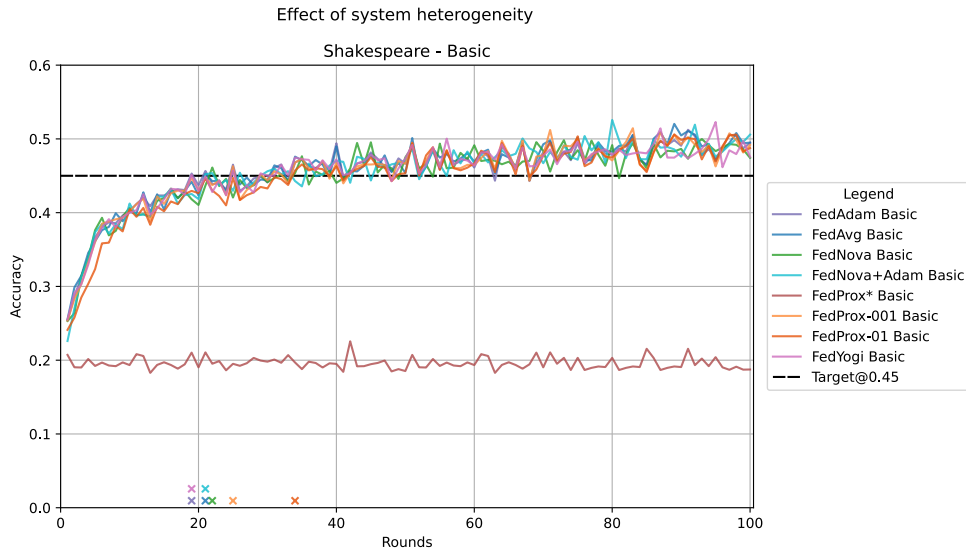


Figure IV.20: Accuracy convergence of algorithms on Shakespeare. Basic scenario.

Regrettably, no algorithm seems to truly outshine the others in our evaluation. Even FedNova, which showed promise in other contexts, doesn't seem to offer a distinct edge here. It's especially noteworthy that the adaptive FedProx fails to improve, at all. Upon closer examination, we see that the value of μ fluctuated between 0.2 and 0.55, introducing substantial regularization. This level of regularization may not be suited for this dataset or for the RNN model we're working with.

Given that this dataset doesn't offer much in the way of new insights, we'll give it a swift overview. Regarding the highest accuracy achieved across the entire test set, as depicted in Figure IV.21, we can see that as anticipated, the algorithms converge to a higher point when faced with stragglers compared to when there are dropouts. Also, FedNova tends to fall a bit behind the others. On the other hand, FedNova+Adam tends to beat the others, but by a tiny margin.

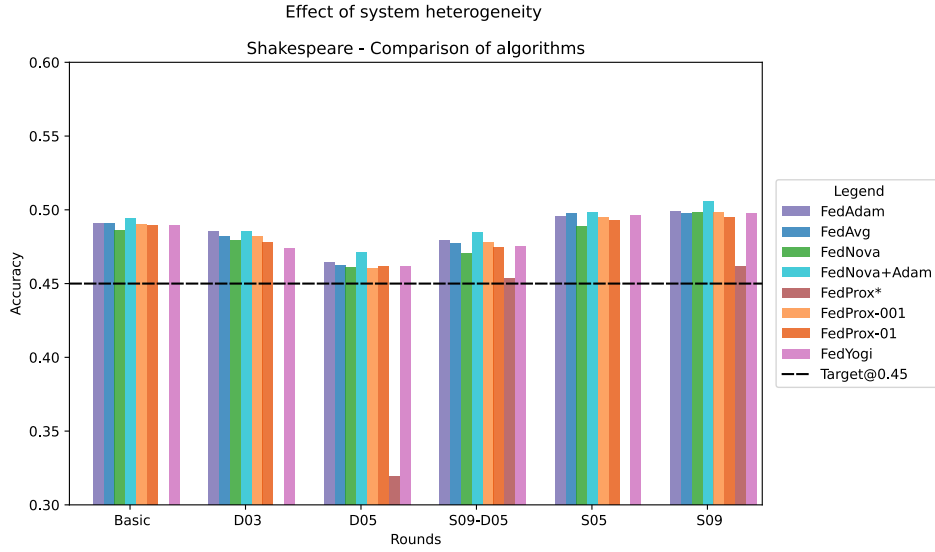


Figure IV.21: Barplot of the best accuracy on the test set for the Shakespeare dataset. 100 rounds.

When discussing fairness, it's interesting to note that this time FedProx shows the lowest variance. Nonetheless, FedNova consistently performs well in terms of the gap. Another metric worth considering is the percentage of clients achieving the threshold on their test data. In the S05 scenario, we find that FedAvg reaches 85.15%, FedNova clocks in at 83.18%, FedNova+Adam at 84.67%, and FedProx (with $\mu = 0.1$) leads slightly with 85.75%. This suggests that while FedNova frequently stands out, other algorithms like FedProx can also be competitive in certain situations.

	Basic	S05	D05
FedAdam	0.7738 / 0.0050	0.6726 / 0.0052	0.7834 / 0.0046
FedAvg	0.8333 / 0.0049	0.7262 / 0.0051	0.7167 / 0.0049
FedNova	0.6587 / 0.0049	0.6333 / 0.0046	0.7976 / 0.0050
FedNova+adam	0.7286 / 0.0055	0.8750 / 0.0056	0.7 / 0.0049
FedProx(mu=0.01)	0.7167 / 0.0049	0.6571 / 0.0044	0.75 / 0.0048
FedProx(mu=0.1)	0.6667 / 0.0048	0.6929 / 0.0044	0.675 / 0.0045
FedYogi	0.8462 / 0.0050	0.6726 / 0.0050	0.6904 / 0.0050

Table IV.5: Comparison of two fairness metrics on the Shakespeare dataset, shown as **gap / variance**. Lower is better. FedProx* omitted as it did not reach the required threshold.

2.4 Extra: fine-tuning and utility of federated learning

Alongside our main experiments, we were keen to dip our toes into the realm of personalized models. To venture into this, we simply introduced an additional step of local training on each client. In a real-world scenario, the final global model will eventually be dispatched to the clients. But what if, upon receiving the model, the client runs it for an additional epoch? The idea is to slightly tailor or "overfit" the model to the client's specific dataset. Let's delve into this using the CIFAR10 dataset ($\alpha_1 = 0.5, \alpha_2 = 0.5$) and focus on FedAvg, as well as our two top contenders, FedNova and FedNova+Adam.

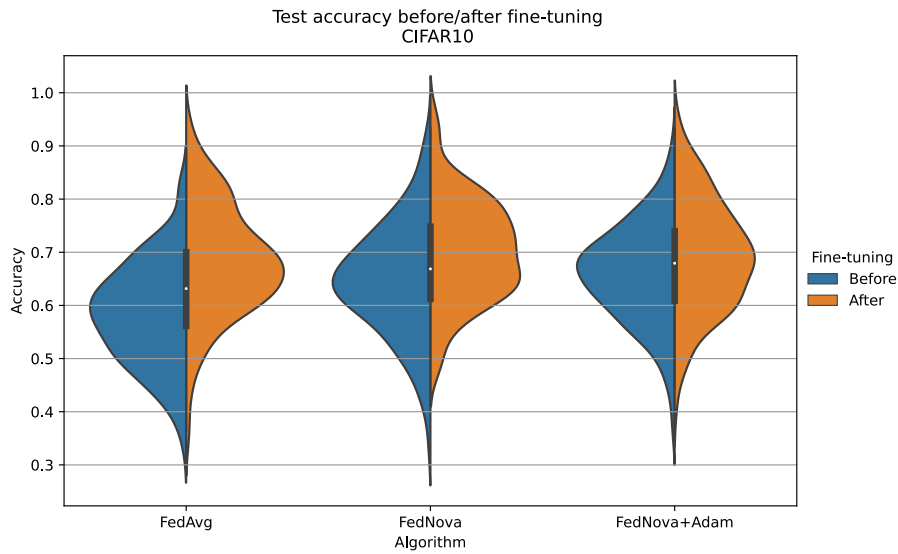


Figure IV.22: Violin plot of the test accuracies for every client, before and after fine-tuning on the CIFAR10 dataset.

From Figure IV.22, several insights emerge. First, it's evident that prior to the fine-tuning phase, the accuracy distribution bears a striking resemblance to a Gaussian curve, especially for FedNova. Moving forward, post the one-epoch fine-tuning, we observe a slight, yet noticeable upward shift in the accuracy distribution, indicative of a positive effect.

“The personalization problem raises the question of how much learning should happen at the local vs global level: if the local models are expressive enough, local learning alone should suffice to learn good models with enough samples per user, yet if the data has some shared structure across users, a global model may help us learn more efficiently.”

– Bietti et al. (2022)

This quote raises a pivotal question: Is Federated Learning even useful if local training is equally or more effective? In an attempt to tackle this question, we conducted an additional experiment on the FEMNIST dataset using FedAvg. Note that the evaluation method plays a significant role in these observations. In our setup, the test dataset quite closely mirrored the training set, as images for both originated from the same sources. While this might introduce a potential bias, especially for clients with limited data who still managed to achieve commendable test results, we believe it somewhat mirrors real-world scenarios.

Our findings indicated that the global model was less beneficial for clients possessing a large volume of data, yet it proved advantageous for those with limited data. However, it's noteworthy that after a single epoch of fine-tuning, federated learning using FedAvg produced superior outcomes compared to exclusive local training. This suggests that federated learning has its merits and can potentially outdo conventional training approaches, with just a bit of local refinement. Nevertheless, to draw definitive conclusions, more extensive research is essential, and this is an avenue we leave for further exploration.

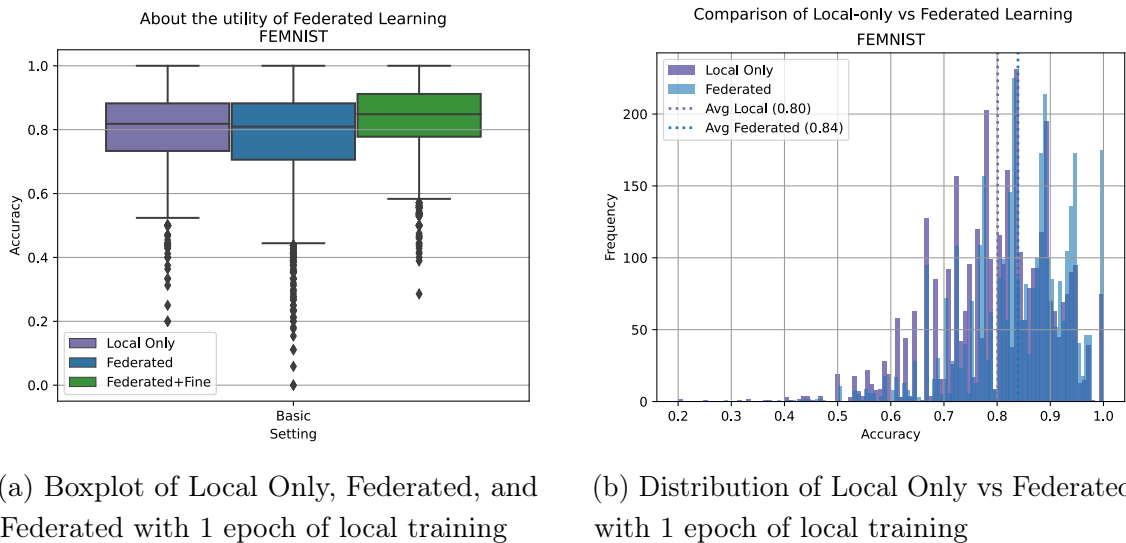


Figure IV.23: Last experiment on FEMNIST about the sole utility of FL

Chapter V

Conclusion

As our journey into the realm of Federated Learning draws to a close, it is evident that this avenue of machine learning holds great potential, offering a solution to the actual problem of data collection. Diving into the world of Federated Learning has been an eye-opening experience, showing us the vast potential of this approach.

Federated Learning, in essence, has made it feasible to train machine learning models in scenarios where traditional methods fall short, respecting user privacy and data sovereignty. However, as with any evolving field, it does not come without its intricacies. A plethora of algorithms exist, each tailored to specific needs, be it faster convergence, resilience to adversities, communication efficiency or fairness.

Through the course of this thesis, we delved deep into the nuances of Federated Learning, investigating three distinct datasets and the suite of algorithms composed of FedAvg, FedProx, FedNova, FedAdam, and FedYogi. One undeniable takeaway is that the concept of creating a fused model by aggregating weights from various local models is effective. By aggregating local model weights, we can successfully train a global model that delivers commendable results.

FedAvg, with its simplicity and effectiveness, really impressed us. Notably straightforward to implement, it surprisingly held its ground against other more complex methods. FedProx, while promising on paper, was a bit of a letdown in practice. Despite its announced resilience in straggler-rich environments, our experiments, in sync with other recent findings, showed its convergence rate lagging behind. Given the added complexity of tuning the μ parameter, the potential gains of using FedProx seem outweighed by the associated challenges. FedNova, on the other hand, emerged as our top recommendation. It consistently matched or outperformed its peers for the FEMNIST dataset without requiring any intricate hyperparameter tweaking. Further, the possibility to combine it with a penalization term in the same way as FedProx makes it an alluring choice. Moreover, it also took the crown in terms of fairness, showing a notable positive impact on the performance of weaker

clients. We were, however, a bit taken aback when we didn't see these same standout results in our Shakespeare experiments. This just goes to show the unique challenges and quirks of the federated learning landscape, and perhaps the specificities of the model used. The momentum-based algorithms, like FedAdam and FedYogi, while effective when paired with FedNova, didn't bring significant improvements on their own.

To explore further the realm of Federated Learning, it would be compelling to evaluate these algorithms in practical scenarios, involving thousands of varied clients under stringent privacy requirements. In such contexts, these algorithms would truly be put to the test, facing obstacles like noise interference, parameter restrictions, and other inherent challenges. Also, our preliminary experiments with fine-tuning suggest an exciting avenue worth exploring: federated learning models tailored not just to individual clients but perhaps to specific client groups or clusters. Such semi-personalized models could harness the power of shared characteristics among client groups while still preserving the nuances that make each client unique.

Wrapping things up, the world of Federated Learning has been nothing short of a thrilling one to navigate. The very idea that we can (almost) seamlessly weave together models trained from scattered sources of data, and to do so with such simplicity, is both empowering and promising for the future of machine learning.

■

Appendix A

Theoretical background

1 Assumptions on optimization

Assumption 1 (Unbiased stochastic gradient).

$$\mathbb{E}_{\xi \sim \mathcal{D}_c} [g_c(\xi, \mathbf{w})] = \nabla F_c(\mathbf{w}) \quad (\text{A.1})$$

Assumption 2 (Bounded Variance).

$$\mathbb{E}_{\xi \sim \mathcal{D}_c} [\|g_c(\xi, \mathbf{w}) - \nabla F_c(\mathbf{w})\|^2] \leq \sigma^2, \quad \forall c \in S^t, \sigma^2 \geq 0. \quad (\text{A.2})$$

Assumption 3 (L-Lipschitz Smoothness).

$$\|\nabla F_c(\mathbf{x}) - \nabla F_c(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \quad \forall c \in S^t \quad (\text{A.3})$$

Assumption 4 (Bounded Dissimilarity). For any set of weights $p_c \geq 0, \sum p_c = 1$, there exist constants $\beta^2 \geq 1, \kappa^2 \geq 0$ such that

$$\sum_{c \in S^t} p_c \|\nabla F_c(\mathbf{w})\|^2 \leq \beta^2 \left\| \sum_{c \in S^t} p_c \nabla F_c(\mathbf{w}) \right\|^2 + \kappa^2. \quad (\text{A.4})$$

If local functions are identical to each other, then $\beta^2 = 1, \kappa^2 = 0$

2 Gradient descent

The stochastic gradient descent (SGD) algorithm is one of the most widely used optimization techniques in deep learning. In this method, the gradient of a loss function with respect to the model parameters is computed for each training example, and then the model parameters are updated in the direction of the negative gradient.

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \partial_{\mathbf{w}} f(\xi, \mathbf{w}^{t-1}) \quad (\text{A.5})$$

The minibatch SGD extends this approach by computing the gradient on a subset of the training data (i.e., a minibatch) rather than on a single training example. This can significantly reduce the variance of the gradient estimates and speed up convergence.

$$\mathbf{g}^t := \mathbf{g}^t(\mathcal{B}) = \partial_{\mathbf{w}} \frac{1}{|\mathcal{B}|} \sum_{\xi \in \mathcal{B}} f(\xi, \mathbf{w}^{t-1}) = \frac{1}{|\mathcal{B}|} \sum_{\xi \in \mathcal{B}} \partial_{\mathbf{w}} f(\xi, \mathbf{w}^{t-1}) \quad (\text{A.6})$$

The minibatch SGD can be expressed as a sum of individual stochastic gradients, computed on each example in the minibatch. The sum is then divided by the minibatch size to obtain an average gradient estimate. To take even more advantage of the variance-reduction effect, one can introduce a momentum term that accumulates past gradients and smooths out the noisy updates.

3 Momentum optimizers

Remark. Much of this section draws inspiration from the excellent book “Dive into Deep Learning” by Zhang et al. (2021). It is included to provide a more comprehensive theoretical backdrop.

The idea of the momentum algorithm is to maintain a moving average of the gradients computed in the previous iterations, weighted by some factor. This moving average is then used to update the model parameters, instead of the instantaneous gradient computed on the current minibatch. By doing so, the momentum algorithm “remembers” the direction of the previous gradients and reduces the impact of noisy updates. The momentum term can be interpreted as a “ball” rolling down the loss function landscape, which accumulates momentum as it moves downhill and avoids getting stuck in narrow valleys. Zhang et al. (2021)

It is a powerful optimization method that can significantly improve the convergence speed and robustness of deep learning models and it is more effective in cases where

the optimization problem is ill-conditioned, i.e., where there are some directions where progress is much slower than in others. It has been extensively studied in the literature, and various variations have been proposed such as Adagrad, RMSProp, and their extensions Adam and Yogi.

Adagrad

In the algorithm proposed by Duchi, Hazan, and Singer (2011), past gradient variance is accumulated in $\mathbf{s}^t$, which is used to calculate individual learning rates per coordinate. The operations are applied coordinate-wise, and η is the learning rate while ϵ is an additive constant that prevents division by zero. Initializing $\mathbf{s}_0$ to zero, we have:

$$\begin{aligned}\mathbf{s}^t &= \mathbf{s}^{t-1} + (\mathbf{g}^t)^2 \\ \tilde{\mathbf{g}}^t &= \mathbf{g}^t \cdot \frac{\eta}{\sqrt{\mathbf{s}^t + \epsilon}} \\ \mathbf{w} &= \mathbf{w}^{t-1} - \tilde{\mathbf{g}}^t\end{aligned}\tag{A.7}$$

The Adagrad algorithm is known to have a predefined schedule for decreasing the learning rate at an effective rate of $\mathcal{O}(t^{-\frac{1}{2}})$. While this is appropriate for convex problems, it may not be ideal for non-convex problems commonly encountered in deep learning. To address this issue, Tieleman, Hinton, et al. (2012) proposed the RMSProp algorithm as a simple solution.

RMSProp

In the Adagrad algorithm, the squares of the gradient $\mathbf{g}^t$ are accumulated into a state vector $\mathbf{s}^t = \mathbf{s}^{t-1} + (\mathbf{g}^t)^2$, which keeps growing without bound due to the lack of normalization. One possible solution is to use $\mathbf{s}^t/t$, but this can take a long time to converge, and the procedure remembers the full trajectory of values. An alternative is to use a leaky average in the same way as the momentum method, i.e.,

$$\begin{aligned}\mathbf{s}^t &= \gamma \mathbf{s}^{t-1} + (1 - \gamma)(\mathbf{g}^t)^2 \\ \tilde{\mathbf{g}}^t &= \mathbf{g}^t \cdot \frac{\eta}{\sqrt{\mathbf{s}^t + \epsilon}} \\ \mathbf{w} &= \mathbf{w}^{t-1} - \tilde{\mathbf{g}}^t\end{aligned}\tag{A.8}$$

for some parameter $\gamma > 0$.

Adam

Adam, by Kingma and Ba (2014) combines the advantages of the two other extensions of stochastic gradient descent: AdaGrad and RMSProp. Unlike RMSProp, which only uses the average of the first moments of the gradients, Adam also makes use of

the average of the second moments of the gradients. This is done by calculating an exponential moving average of both the gradient and the squared gradient, with the decay rates controlled by the parameters β_1 and β_2 .

$$\begin{aligned}\mathbf{v}^t &\leftarrow \beta_1 \mathbf{v}^{t-1} + (1 - \beta_1) \mathbf{g}^t \\ \mathbf{s}^t &\leftarrow \beta_2 \mathbf{s}^{t-1} + (1 - \beta_2) (\mathbf{g}^t)^2\end{aligned}\tag{A.9}$$

One can leverage the property $\sum_{i=0}^t \beta^i = \frac{1-\beta^{t+1}}{1-\beta}$ to re-normalize the terms. By re-scaling the first and second moments estimates, we can prevent them from becoming too small or too large. Specifically, to correct for the bias introduced by the initial values of these moving averages, Adam first calculates biased estimates before calculating bias-corrected estimates, as follows:

$$\hat{\mathbf{v}}^t = \frac{\mathbf{v}^t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}^t = \frac{\mathbf{s}^t}{1 - \beta_2^t}\tag{A.10}$$

After obtaining the estimates for the second moment $\hat{\mathbf{s}}^t$ and momentum $\hat{\mathbf{v}}^t$, we can use them to compute the update equation. First, we rescale the gradient similar to RMSProp by dividing it by the square root of the second moment estimate $\hat{\mathbf{s}}^t$ plus a small constant ϵ , and then multiply by the learning rate η and the momentum $\hat{\mathbf{v}}^t$ as:

$$\tilde{\mathbf{g}}^t = \frac{\eta \hat{\mathbf{v}}^t}{\sqrt{\hat{\mathbf{s}}^t} + \epsilon}\tag{A.11}$$

While this rescaling is similar to RMSProp, Adam uses the momentum estimate instead of the gradient itself. Additionally, the rescaling is done using the square root of the estimate only, leaving the ϵ outside, which has been shown to work better in practice. A value of $\epsilon = 10^{-6}$ is commonly used for numerical stability. Finally, we can compute the update equation by subtracting the rescaled gradient from the current weights:

$$\mathbf{w} \leftarrow \mathbf{w}^{t-1} - \tilde{\mathbf{g}}^t\tag{A.12}$$

This completes one iteration of the Adam optimization algorithm.

Yogi

Despite its popularity, Adam has been shown to diverge due to poor variance control in some situations. This issue arises when the second moment estimate in $\mathbf{s}^t$ blows up. To address this issue, a hotfix for $\mathbf{s}^t$ was proposed by Zaheer et al. (2018) that replaces the update rule for the second moment in Adam with a conditional update

that adapts based on the sign of the change in gradients. The update rules for the first and second moments in Yogi are as follows:

$$\begin{aligned} \mathbf{v}^t &\leftarrow \beta_1 \mathbf{v}^{t-1} + (1 - \beta_1) \mathbf{g}^t \\ \mathbf{s}^t &\leftarrow \mathbf{s}^{t-1} - (1 - \beta_2) \text{sign}(\mathbf{s}^{t-1} - (\mathbf{g}^t)^2) \cdot (\mathbf{g}^t)^2 \end{aligned} \tag{A.13}$$

The rest of the update equation is analogous to Adam, using the same rescaling and bias correction techniques. By selectively updating the second moment estimate, Yogi can maintain better control over the variance of the gradient estimates, leading to improved robustness in certain scenarios where Adam may diverge.

Appendix B

Details

1 Algorithms

1.1 FedProx*

The server takes care of the adaptability feature in the FedProx* algorithm. We calculate the average validation accuracies for the clients and compares it with the last two rounds. If the accuracy at $t - 2$ is higher than the accuracy at t we increase μ by $0.05 \times \eta_s$. If not, we do the opposite. Multiplying by the server's learning rate ensures that μ adjusts in finer steps as the training continues.

1.2 FedAdadelta

Algorithm 7 AdaDelta as the momentum server optimizer

Functions of the framework

```

1 AGGREGATION(*):
2    $\Delta^t = \sum_{c \in S^t} p_c \Delta_c^t$ 
3    $m^t = \beta_1 m^{t-1} + (1 - \beta_1) \Delta_t^2$ 
4   return  $(\Delta^t, m^t)$ 
5
6 SERVEROPT(*):
7    $\tilde{\Delta}^t = \Delta^t \sqrt{v^{t-1} + \epsilon} / \sqrt{m^t + \epsilon}$ 
8    $v^t = \beta_1 v^{t-1} + (1 - \beta_1) \tilde{\Delta}_t^2$ 
9   return  $\mathbf{w}^t + \eta_s \tilde{\Delta}^t$ 
```

2 Tables

2.1 Resnet-18 architecture

The architecture is identical to the one presented by Reddi et al. (2020), where batch normalization (BN) layers are substituted with group normalization (GN) layers.

Layer	Output Shape	Parameters
- Conv2d	[16, 64, 12, 12]	9,408
- GroupNorm	[16, 64, 12, 12]	128
- ReLU	[16, 64, 12, 12]	–
- MaxPool2d	[16, 64, 6, 6]	–
- Sequential	[16, 64, 6, 6]	–
- BasicBlock	[16, 64, 6, 6]	73,984
- BasicBlock	[16, 64, 6, 6]	73,984
- Sequential	[16, 128, 3, 3]	–
- BasicBlock	[16, 128, 3, 3]	230,144
- BasicBlock	[16, 128, 3, 3]	295,424
- Sequential	[16, 256, 2, 2]	–
- BasicBlock	[16, 256, 2, 2]	919,040
- BasicBlock	[16, 256, 2, 2]	1,180,672
- Sequential	[16, 512, 1, 1]	–
- BasicBlock	[16, 512, 1, 1]	3,673,088
- BasicBlock	[16, 512, 1, 1]	4,720,640
- AdaptiveAvgPool2d	[16, 512, 1, 1]	–
- Linear	[16, 10]	5,130
Total params		11,181,642

Table B.1: ResNet model structure (with CIFAR10 input, batch of 16). The model parameters takes around 45MB.

2.2 TwoCNN architecture

Layer	Output shape	Parameters
- Sequential	[16, 64, 6, 6]	–
- Conv2d	[16, 32, 22, 22]	2,432
- ReLU	[16, 32, 22, 22]	–
- MaxPool2d	[16, 32, 12, 12]	–
- Conv2d	[16, 64, 10, 10]	51,264
- ReLU	[16, 64, 10, 10]	–
- MaxPool2d	[16, 64, 6, 6]	–
- Sequential	[16, 10]	–
- AdaptiveAvgPool2d	[16, 64, 7, 7]	–
- Flatten	[16, 3136]	–
- Linear	[16, 512]	1,606,144
- ReLU	[16, 512]	–
- Linear	[16, 10]	5,130
Total params		1,664,970

Table B.2: TwoCNN model structure (with CIFAR10 input, batch of 16, hidden_size=32). The model parameters takes around 7MB.

2.3 NextCharLSTM architecture

Layer	Output shape	Parameters
- NextCharLSTM	[8, 80]	–
- Sequential	[8, 80, 256]	–
- Embedding	[8, 80, 8]	640
- LSTM	[8, 80, 256]	798,720
- Linear	[8, 80]	20,560
Total params		819,920

Table B.3: NextCharLSTM model structure (with Shakespeare data, batch of 8). The model parameters takes around 3MB.

2.4 CIFAR10 Resnet - Fairness - Data Heterogeneity

	0.5-0.5	0.5-inf	inf-0.5	inf-inf
FedAdam	6 (0.58)	1 (0.52)	6 (0.65)	3 (0.32)
FedAvg	5 (0.57)	2 (0.54)	4 (0.56)	7 (0.38)
FedNova	4 (0.49)	5 (0.64)	2 (0.61)	1 (0.34)
FedProx(mu=0.01)	7 (0.58)	3 (0.54)	5 (0.55)	4 (0.38)
FedProx(mu=0.1)	3 (0.55)	4 (0.58)	3 (0.53)	6 (0.32)
FedProx*(mu=adaptive)	2 (0.57)	7 (0.52)	7 (0.56)	5 (0.34)
FedYogi	1 (0.51)	6 (0.56)	1 (0.47)	2 (0.36)

Table B.4: Rank by variance in terms of full test set. Ranks are by column. Values in parenthesis correspond to the difference between the max and min test accuracy between clients. Lower is better.

3 Figures

3.1 Flower framework

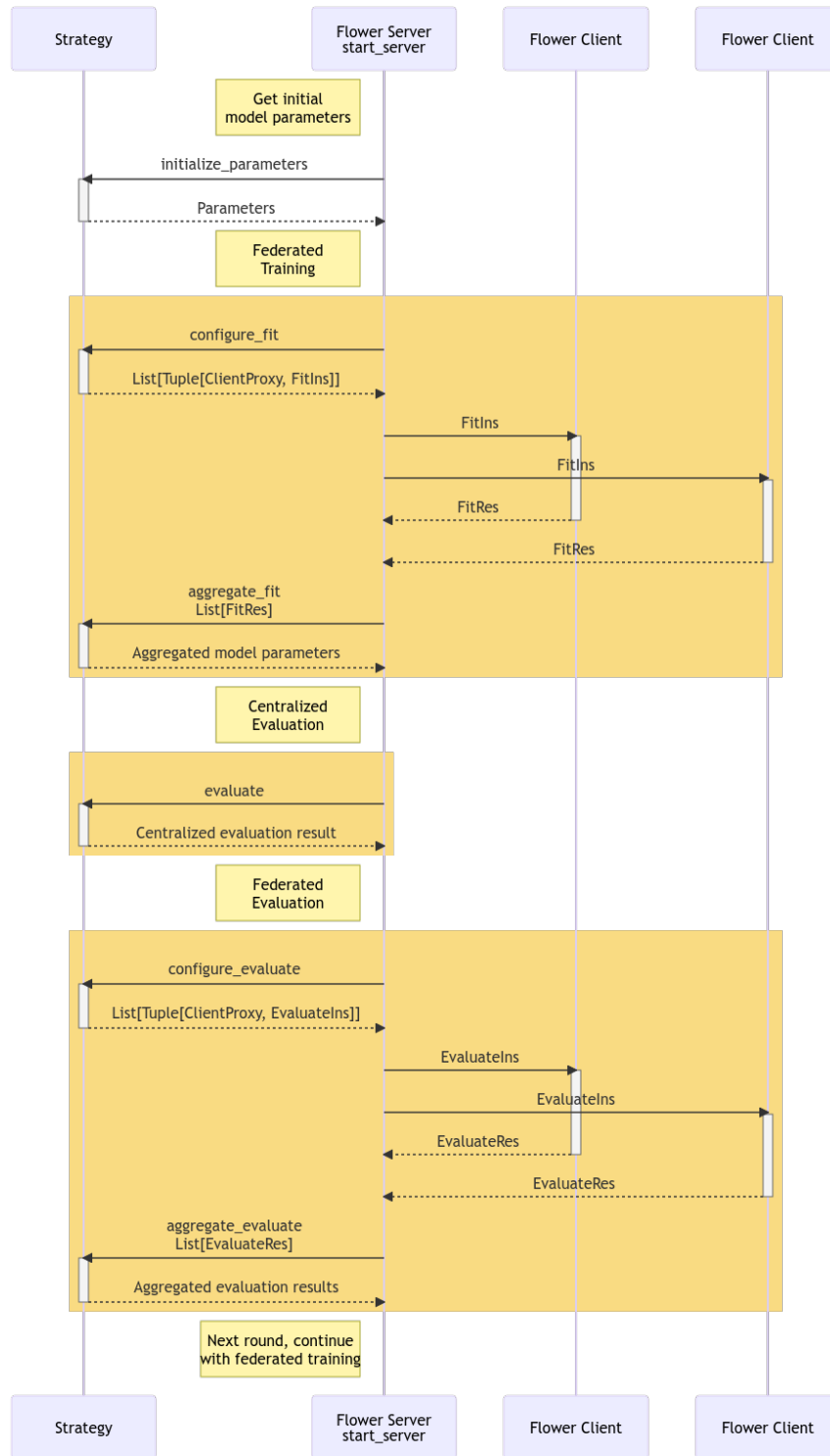


Figure B.1: Diagram of the Flower process with 2 users

3.2 CIFAR10 - Individual accuracies on validation set

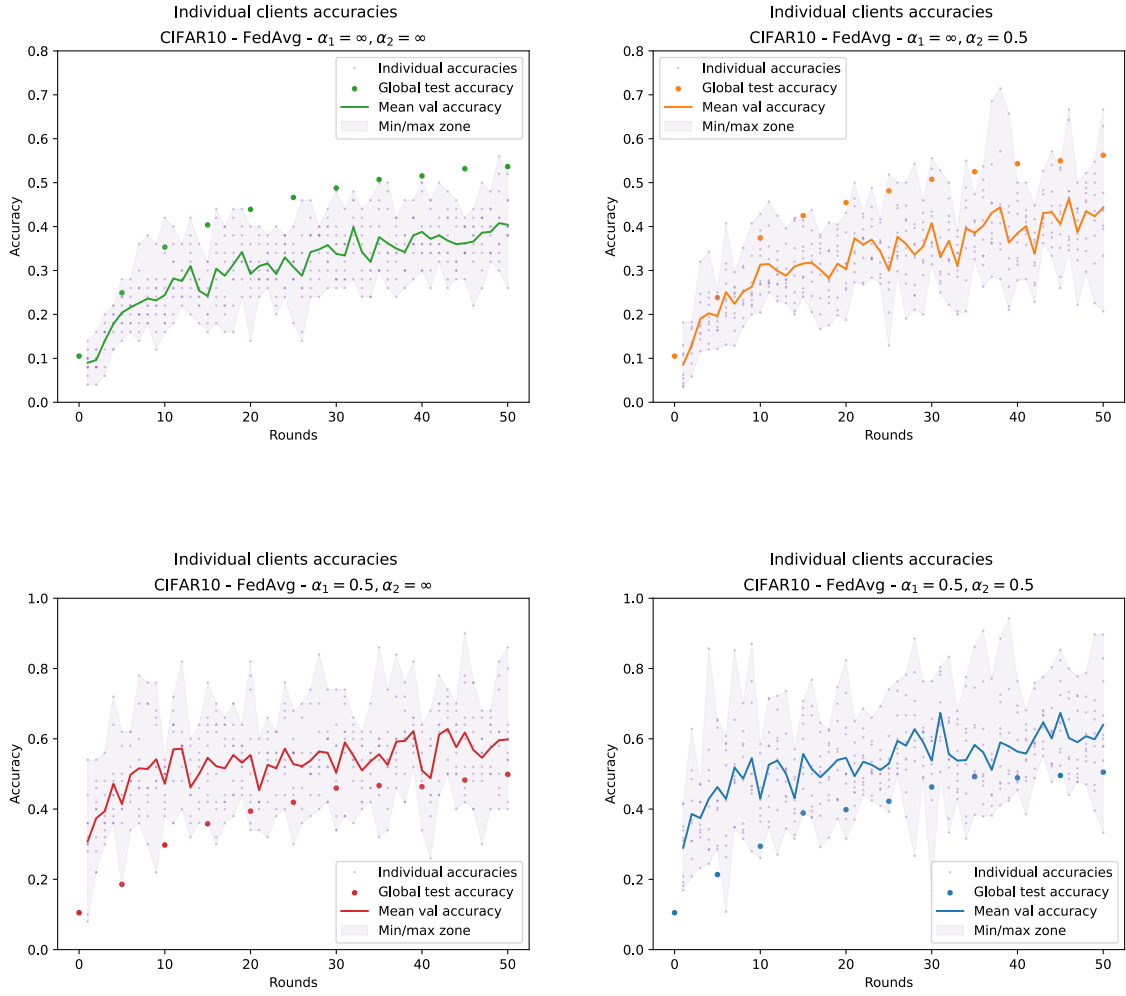


Figure B.2: Test accuracies of selected clients for training. Note that clients are not the same between rounds. The accuracy measure is done after local training in order to see the overfitting. Model is ResNet-18.

3.3 CIFAR10 - Resnet - Convergence speed comparison

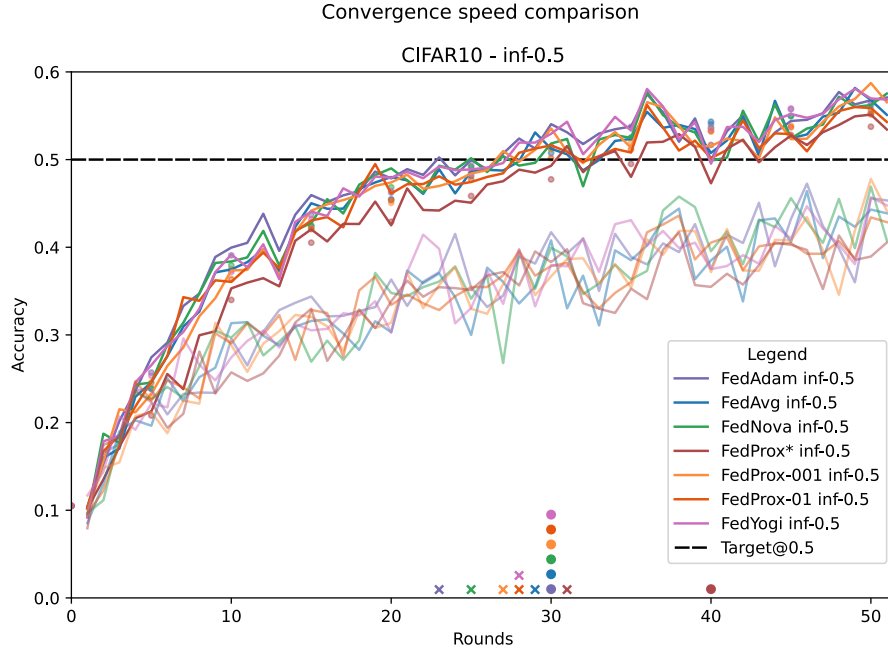


Figure B.3: Comparison of the convergence speed in terms of test accuracy, ResNet-18, CIFAR10 (inf-0.5) dataset. Crosses and dots at the bottom indicates when the algorithms goes above the target for the distributed and full test sets, respectively. Accuracy on the validation set of the clients is shown in transparency.

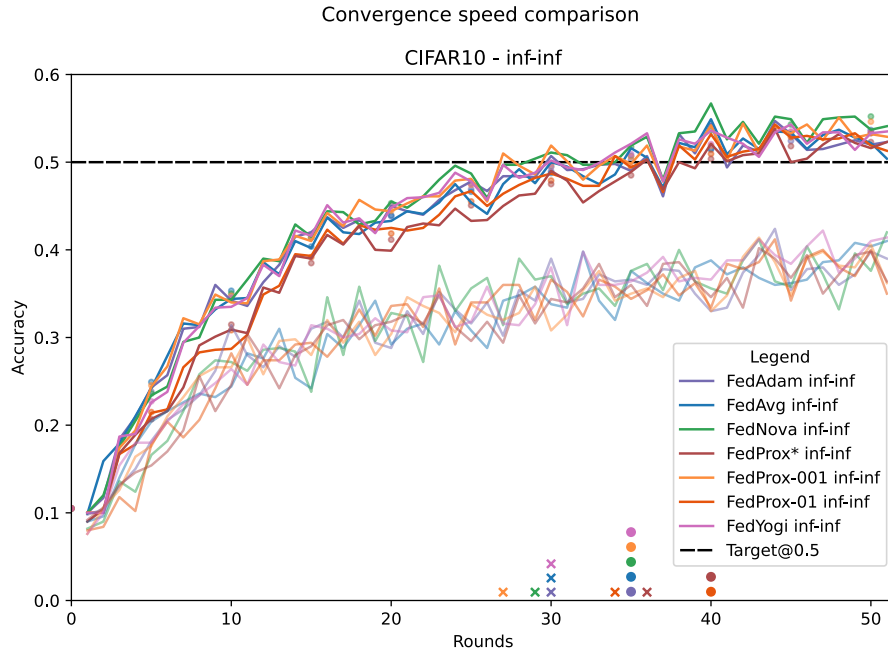


Figure B.4: Comparison of the convergence speed in terms of test accuracy (inf-inf).

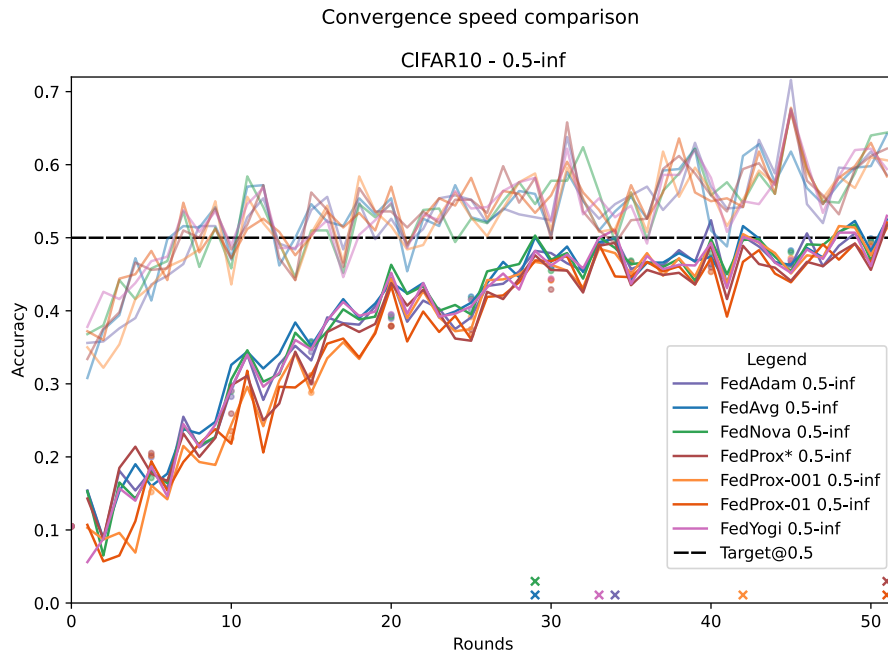


Figure B.5: Comparison of the convergence speed in terms of test accuracy (0.5-inf).

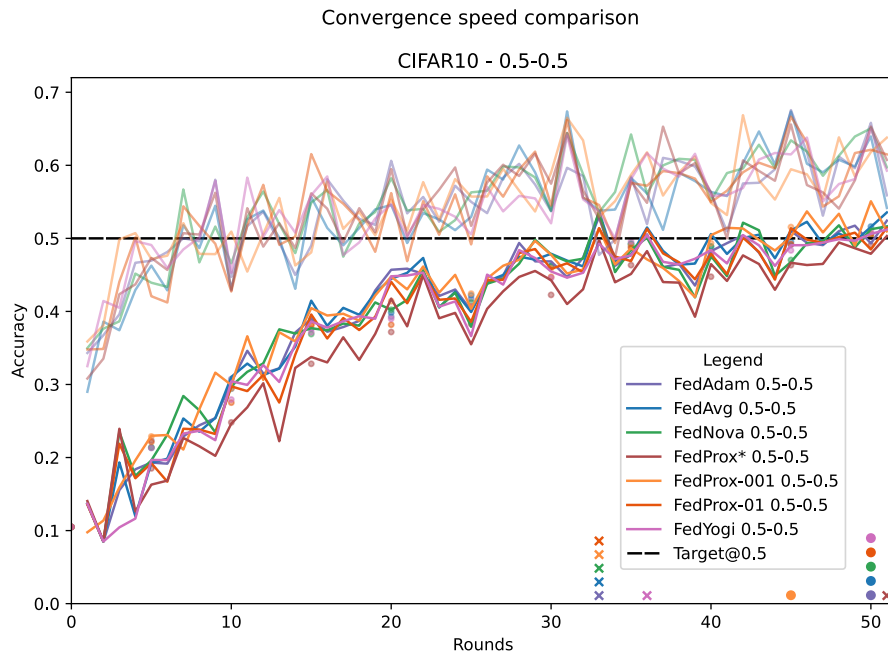


Figure B.6: Comparison of the convergence speed in terms of test accuracy (0.5-0.5).

3.4 CIFAR10 - Barplot comparison of models

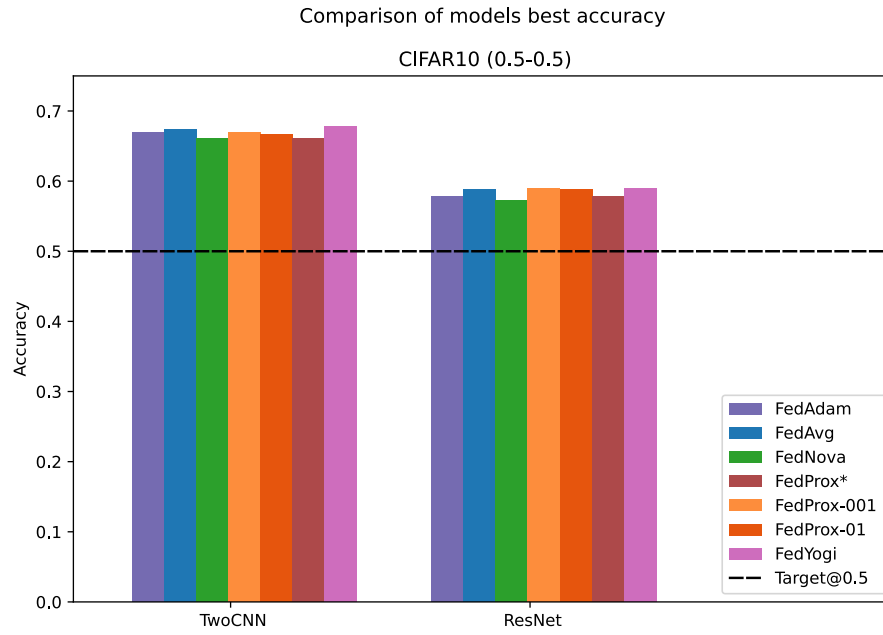


Figure B.7: Comparison of the best full test set accuracies on CIFAR10 for both models. 100 rounds.

3.5 FEMNIST - System heterogeneity - Comparison

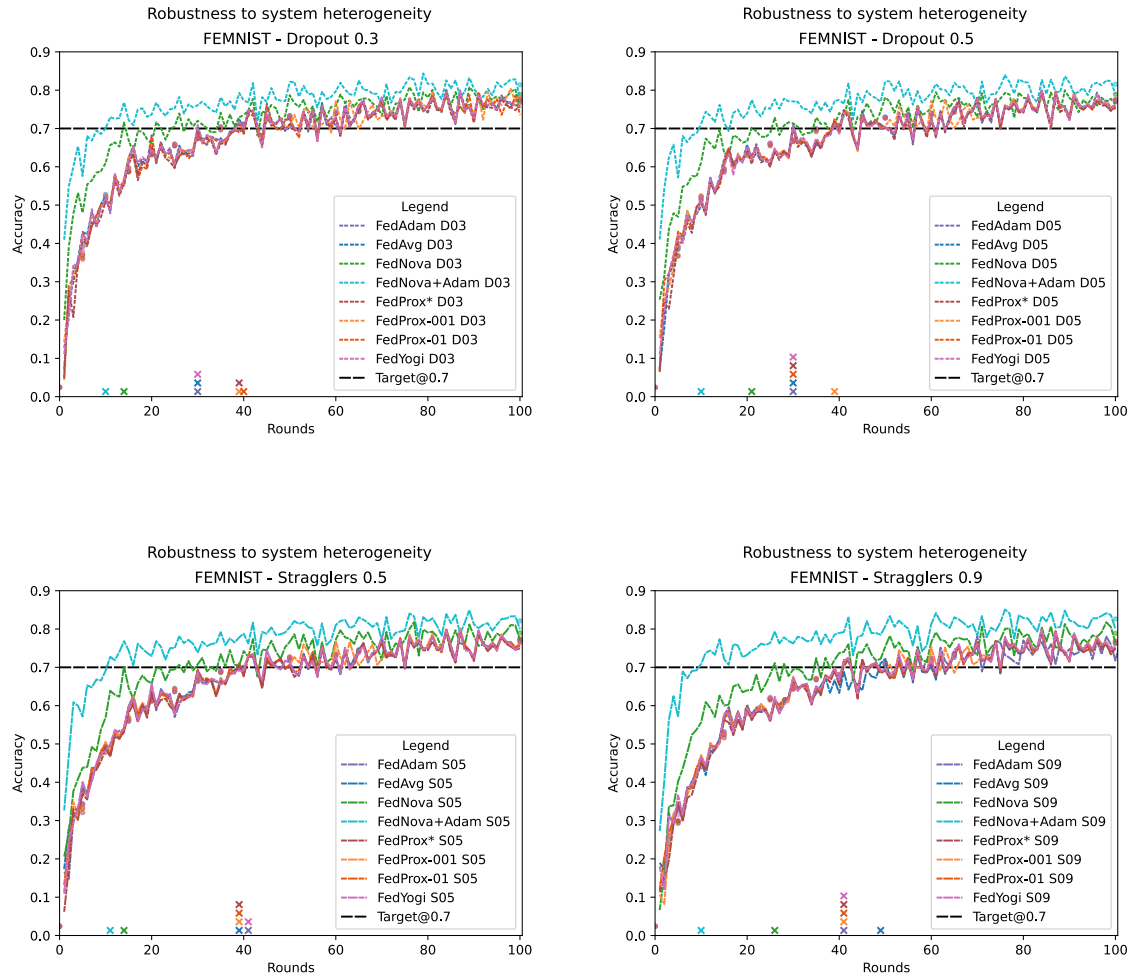


Figure B.8: Client heterogeneity scenarios, algorithms accuracy convergence comparison

3.6 FEMNIST - Fairness histograms - S09D05

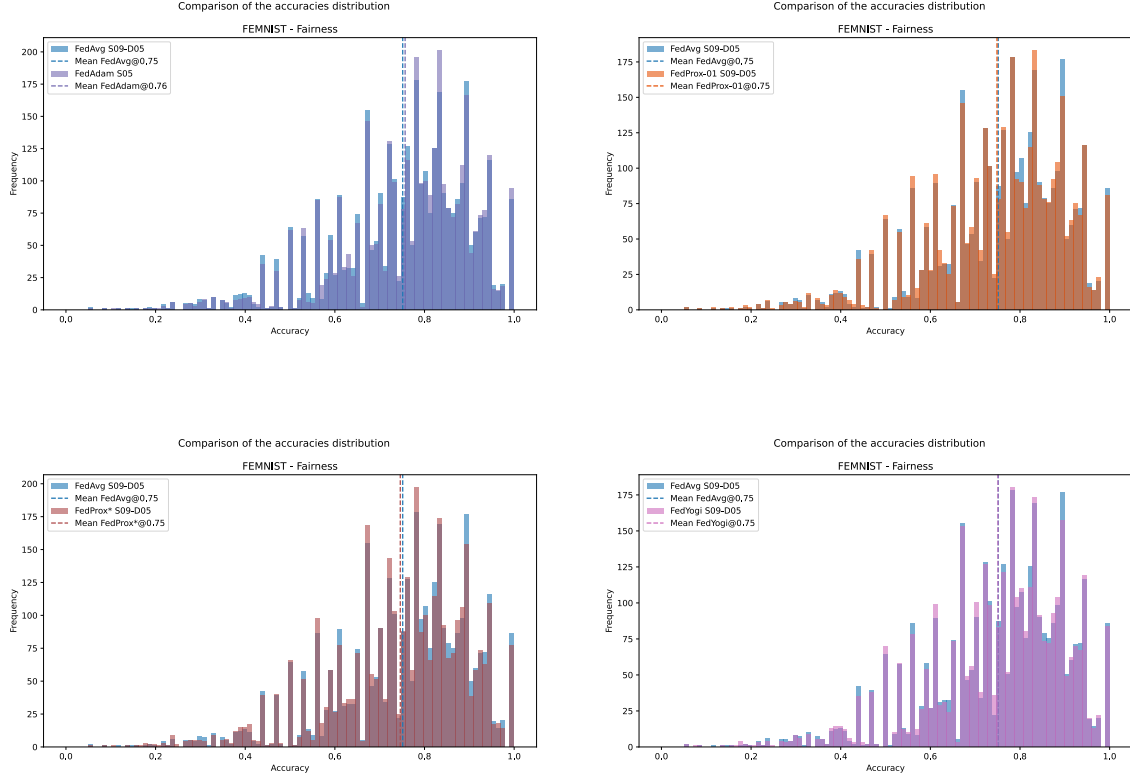


Figure B.9: Fairness histogram comparing FedAvg to other algorithms

3.7 Shakespeare - System heterogeneity - Comparison

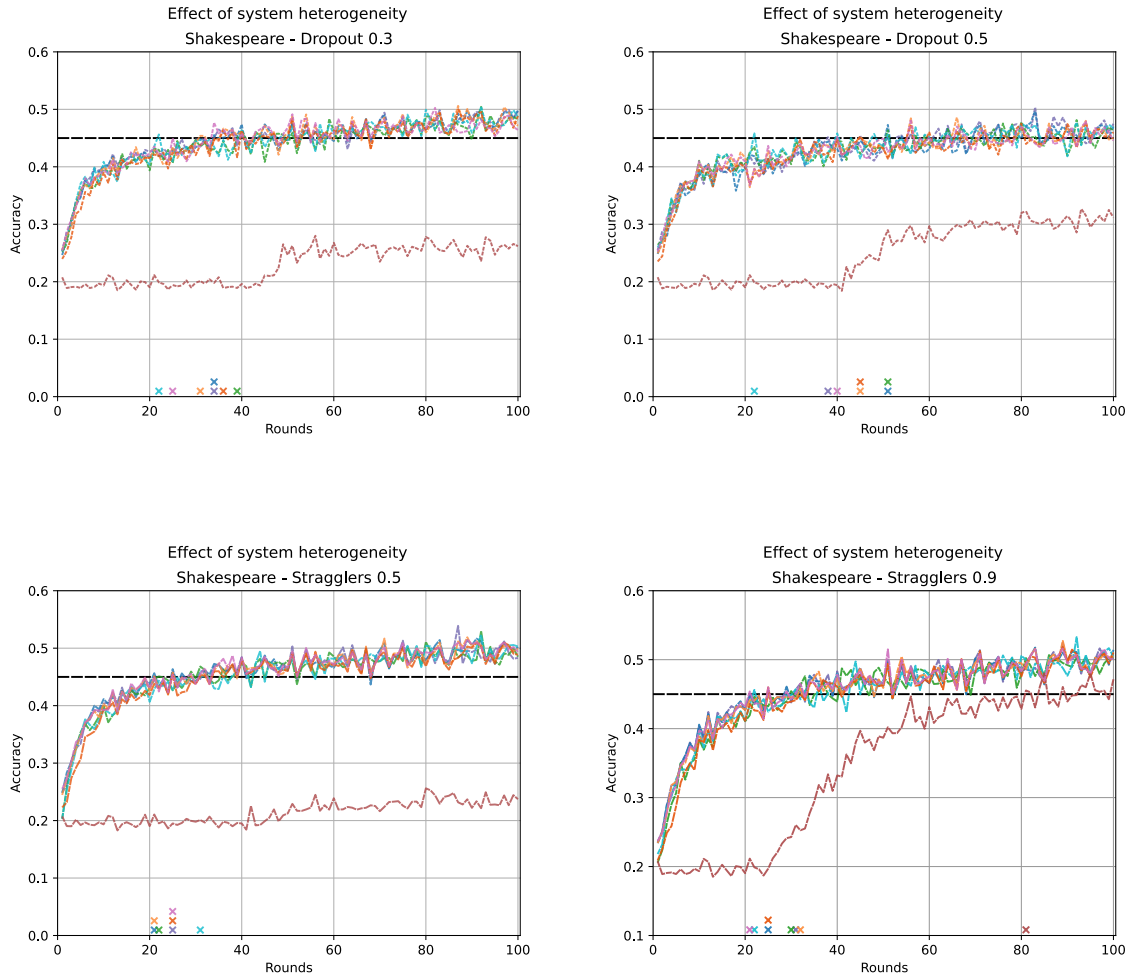


Figure B.10: Client heterogeneity scenarios, algorithms accuracy convergence comparison

3.8 Shakespeare - Fairness histograms - Basic

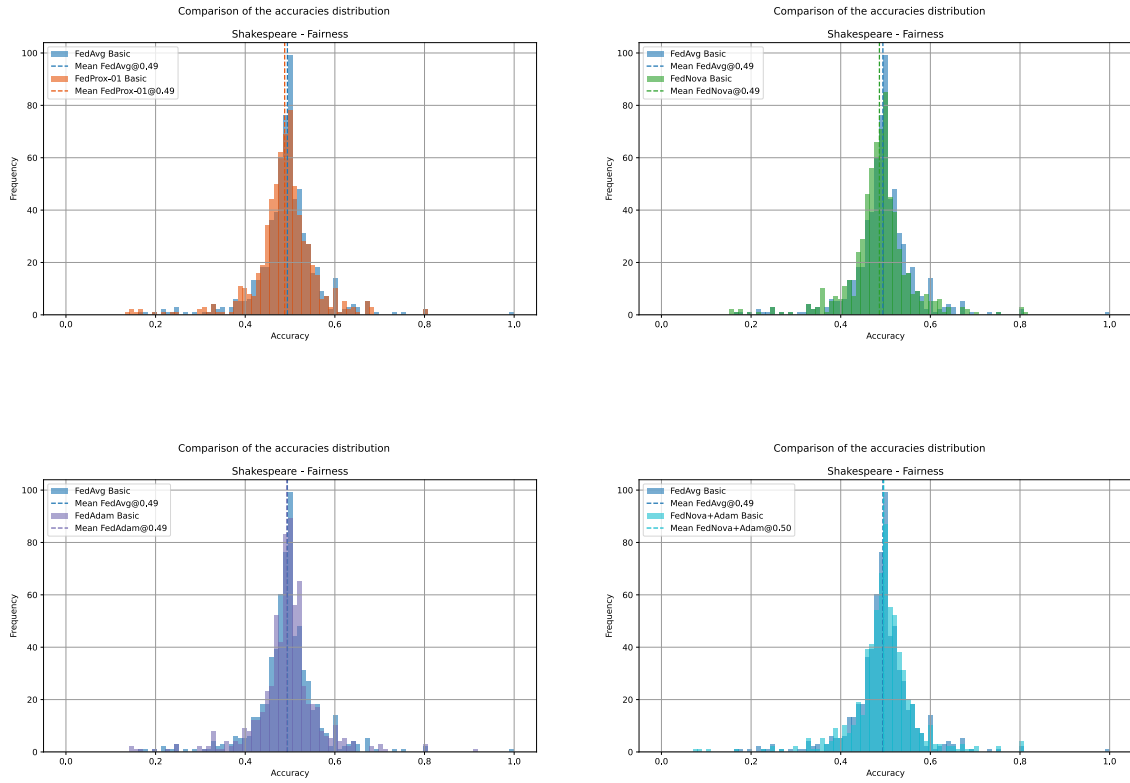


Figure B.11: Fairness histograms, shakespeare, basic setting

Bibliography

- [1] Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. “Differentially Private Learning with Adaptive Clipping”. In: (2022). arXiv: 1905.03871 [cs.LG] (pages 16, 32).
- [2] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. “Secure Single-Server Aggregation with (Poly)Logarithmic Overhead”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1253–1269. ISBN: 9781450370899. DOI: 10.1145/3372297.3417885. URL: <https://doi.org/10.1145/3372297.3417885> (page 16).
- [3] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. “Flower: A Friendly Federated Learning Research Framework”. In: (2020). DOI: 10.48550/ARXIV.2007.14390. URL: <https://arxiv.org/abs/2007.14390> (pages 14, 38).
- [4] Alberto Bietti, Chen-Yu Wei, Miroslav Dudík, John Langford, and Zhiwei Steven Wu. *Personalization Improves Privacy-Accuracy Tradeoffs in Federated Learning*. 2022. arXiv: 2202.05318 [stat.ML] (page 55).
- [5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roseland. “Towards Federated Learning at Scale: System Design”. In: (2019). DOI: 10.48550/ARXIV.1902.01046. URL: <https://arxiv.org/abs/1902.01046> (pages 3, 5).
- [6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. “Practical Secure Aggregation for Federated Learning on User-Held Data”. In: (2016). DOI: 10.48550/ARXIV.1611.04482. URL: <https://arxiv.org/abs/1611.04482> (page 16).

- [7] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. “Federated learning of predictive models from federated Electronic Health Records”. In: *International Journal of Medical Informatics* 112 (2018), pp. 59–67. ISSN: 1386-5056. DOI: <https://doi.org/10.1016/j.ijmedinf.2018.01.007>. URL: <https://www.sciencedirect.com/science/article/pii/S138650561830008X> (page 2).
- [8] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. “LEAF: A Benchmark for Federated Settings”. In: (2018). DOI: 10.48550/ARXIV.1812.01097. URL: <https://arxiv.org/abs/1812.01097> (page 48).
- [9] John Chen, Cameron Wolfe, Zhao Li, and Anastasios Kyrillidis. *Demon: Improved Neural Network Training with Momentum Decay*. 2021. arXiv: 1910.04952 [cs.LG] (page 40).
- [10] Yiqiang Chen, Xiaodong Yang, Xin Qin, Han Yu, Piu Chan, and Zhiqi Shen. “Dealing with Label Quality Disparity in Federated Learning”. In: *Federated Learning: Privacy and Incentive*. Ed. by Qiang Yang, Lixin Fan, and Han Yu. Cham: Springer International Publishing, 2020, pp. 108–121. ISBN: 978-3-030-63076-8. DOI: 10.1007/978-3-030-63076-8_8. URL: https://doi.org/10.1007/978-3-030-63076-8_8 (page 14).
- [11] Dong Chu, Wael Jaafar, and Halim Yanikomeroglu. “On the Design of Communication-Efficient Federated Learning for Health Monitoring”. In: (2022). DOI: 10.48550/ARXIV.2211.16952. URL: <https://arxiv.org/abs/2211.16952> (page 2).
- [12] Yanbo Dai and Songze Li. *Chameleon: Adapting to Peer Images for Planting Durable Backdoors in Federated Learning*. 2023. arXiv: 2304.12961 [cs.LG] (page 15).
- [13] Trung Kien Dang, Xiang Lan, Jianshu Weng, and Mengling Feng. “Federated Learning for Electronic Health Records”. In: *ACM Trans. Intell. Syst. Technol.* 13.5 (June 2022). ISSN: 2157-6904. DOI: 10.1145/3514500. URL: <https://doi.org/10.1145/3514500> (page 2).
- [14] Emiliano De Cristofaro and Gene Tsudik. “Practical private set intersection protocols with linear complexity”. In: *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*. Springer. 2010, pp. 143–159 (page 9).
- [15] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine*

- Learning Research* 12.61 (2011), pp. 2121–2159. URL: <http://jmlr.org/papers/v12/duchi11a.html> (page 61).
- [16] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. “Efficient Private Matching and Set Intersection”. In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–19. ISBN: 978-3-540-24676-3 (page 9).
- [17] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. “Federated Learning for Mobile Keyboard Prediction”. In: (2018). DOI: 10.48550/ARXIV.1811.03604. URL: <https://arxiv.org/abs/1811.03604> (page 2).
- [18] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. “FedML: A Research Library and Benchmark for Federated Machine Learning”. In: (2020). arXiv: 2007.13518 [cs.LG] (page 3).
- [19] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. “Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification”. In: (2019). arXiv: 1909.06335 [cs.LG] (page 35).
- [20] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. “Communication-Efficient Distributed Dual Coordinate Ascent”. In: (2014). DOI: 10.48550/ARXIV.1409.1458. URL: <https://arxiv.org/abs/1409.1458> (page 5).
- [21] Swanand Kadhe, Nived Rajaraman, O. Ozan Koyluoglu, and Kannan Ramchandran. “FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning”. In: (2020). DOI: 10.48550/ARXIV.2009.11248. URL: <https://arxiv.org/abs/2009.11248> (page 16).
- [22] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. “Advances and open problems in federated learning”. In: *arXiv preprint arXiv:1912.04977v3* (2021) (pages 6, 7, 12, 38).
- [23] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2014). DOI: 10.48550/ARXIV.1412.6980. URL: <https://arxiv.org/abs/1412.6980> (pages 30, 61).

- [24] Jakub Konečný, Brendan McMahan, and Daniel Ramage. “Federated Optimization: Distributed Optimization Beyond the Datacenter”. In: (2015). DOI: 10.48550/ARXIV.1511.03575. URL: <https://arxiv.org/abs/1511.03575> (pages 1, 5).
- [25] Raed Kontar, Naichen Shi, Xubo Yue, Seokhyun Chung, Eunshin Byon, Mosharaf Chowdhury, Judy Jin, Wissam Kontar, Neda Masoud, Maher Noueihed, Chinedum E. Okwudire, Garvesh Raskutti, Romesh Saigal, Karandeep Singh, and Zhisheng Ye. “The Internet of Federated Things (IoFT): A Vision for the Future and In-depth Survey of Data-driven Approaches for Federated Learning”. In: (2021). DOI: 10.48550/ARXIV.2111.05326. URL: <https://arxiv.org/abs/2111.05326> (pages 6, 24, 38).
- [26] A. Krizhevsky and G. Hinton. “Learning multiple layers of features from tiny images”. In: *Master’s thesis, Department of Computer Science, University of Toronto* (2009) (page 34).
- [27] Jason D. Lee, Qihang Lin, Tengyu Ma, and Tianbao Yang. “Distributed Stochastic Variance Reduced Gradient Methods and A Lower Bound for Communication Complexity”. In: (2015). DOI: 10.48550/ARXIV.1507.07595. URL: <https://arxiv.org/abs/1507.07595> (page 5).
- [28] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. “A survey on federated learning systems: vision, hype and reality for data privacy and protection”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021) (pages 10, 25).
- [29] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. *Ditto: Fair and Robust Federated Learning Through Personalization*. 2021. arXiv: 2012.04221 [cs.LG] (page 15).
- [30] Tian Li, Anit Kumar Sahu, Ameet S. Talwalkar, and Virginia Smith. “Federated Learning: Challenges, Methods, and Future Directions”. In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 50–60. DOI: 10.1109/msp.2020.2975749 (page 8).
- [31] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. “Federated Optimization in Heterogeneous Networks”. In: (2018). DOI: 10.48550/ARXIV.1812.06127. URL: <https://arxiv.org/abs/1812.06127> (pages 6, 22–25).
- [32] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. “On the Convergence of FedAvg on Non-IID Data”. In: (2019). DOI: 10.48550/ARXIV.1907.02189. URL: <https://arxiv.org/abs/1907.02189> (page 41).

- [33] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. “A Secure Federated Transfer Learning Framework”. In: *IEEE Intelligent Systems* 35.4 (July 2020), pp. 70–82. DOI: 10.1109/mis.2020.2988525. URL: <https://doi.org/10.1109> (page 9).
- [34] Lingjuan Lyu, Han Yu, Jun Zhao, and Qiang Yang. “Threats to Federated Learning”. In: *Federated Learning: Privacy and Incentive*. Ed. by Qiang Yang, Lixin Fan, and Han Yu. Cham: Springer International Publishing, 2020, pp. 3–16. ISBN: 978-3-030-63076-8. DOI: 10.1007/978-3-030-63076-8_1 (page 15).
- [35] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: (2016). DOI: 10.48550/ARXIV.1602.05629. URL: <https://arxiv.org/abs/1602.05629> (pages 5, 20).
- [36] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. “Learning Differentially Private Recurrent Language Models”. In: (2018). arXiv: 1710.06963 [cs.LG] (pages 16, 31, 32).
- [37] Ilya Mironov. “Rényi Differential Privacy”. In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, Aug. 2017. DOI: 10.1109/csf.2017.11. URL: <https://doi.org/10.1109> (page 32).
- [38] Aritra Mitra, Rayana Jaafar, George J. Pappas, and Hamed Hassani. “Linear Convergence in Federated Learning: Tackling Client Heterogeneity and Sparse Gradients”. In: (2021). DOI: 10.48550/ARXIV.2102.07053. URL: <https://arxiv.org/abs/2102.07053> (page 13).
- [39] Milad Nasr, Reza Shokri, and Amir Houmansadr. “Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019. DOI: 10.1109/sp.2019.00065. URL: <https://doi.org/10.1109%2Fsp.2019.00065> (page 15).
- [40] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. “Adaptive Federated Optimization”. In: (2020). DOI: 10.48550/ARXIV.2003.00295. URL: <https://arxiv.org/abs/2003.00295> (pages 18, 20, 30, 35, 40, 46, 65).
- [41] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. “Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity”. In: (2020). DOI: 10.48550/ARXIV.2012.14453. URL: <https://arxiv.org/abs/2012.14453> (page 13).

- [42] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Merouane Debbah. “Federated Learning for Ultra-Reliable Low-Latency V2V Communications”. In: (2018). DOI: 10.48550/ARXIV.1805.09253. URL: <https://arxiv.org/abs/1805.09253> (page 2).
- [43] Ohad Shamir, Nathan Srebro, and Tong Zhang. “Communication Efficient Distributed Optimization using an Approximate Newton-type Method”. In: (2013). DOI: 10.48550/ARXIV.1312.7853. URL: <https://arxiv.org/abs/1312.7853> (page 5).
- [44] Jinhyun So, Basak Guler, and A. Salman Avestimehr. “Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning”. In: (2020). DOI: 10.48550/ARXIV.2002.04156. URL: <https://arxiv.org/abs/2002.04156> (page 16).
- [45] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E. Ali, Basak Guler, and Salman Avestimehr. “LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning”. In: (2021). DOI: 10.48550/ARXIV.2109.14236. URL: <https://arxiv.org/abs/2109.14236> (page 16).
- [46] Google Cloud Tech. *What is Federated Learning*. Feb. 2021. URL: <https://www.youtube.com/watch?v=X8YYWuntt0Y> (page 11).
- [47] Tijmen Tieleman, Geoffrey Hinton, et al. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning 4.2* (2012), pp. 26–31 (page 61).
- [48] Mark Vero, Mislav Balunović, Dimitar I. Dimitrov, and Martin Vechev. *Data Leakage in Tabular Federated Learning*. 2022. arXiv: 2210.01785 [cs.LG] (page 15).
- [49] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horvath, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konecny, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtarik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng,

- Chen Zhu, and Wennan Zhu. “A Field Guide to Federated Optimization”. In: (2021). arXiv: 2107.06917 [cs.LG] (pages 17, 18, 23, 24, 29).
- [50] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. “Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization”. In: (2020). DOI: 10.48550/ARXIV.2007.07481. URL: <https://arxiv.org/abs/2007.07481> (pages 3, 26, 27).
- [51] Yu-Xiang Wang, Borja Balle, and Shiva Kasiviswanathan. “Subsampled Rényi Differential Privacy and Analytical Moments Accountant”. In: (2018). arXiv: 1808.00087 [cs.LG] (page 32).
- [52] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. *Federated Machine Learning: Concept and Applications*. 2019. arXiv: 1902.04885 [cs.AI] (page 9).
- [53] Seongjun Yang, Hyeonji Hwang, Daeyoung Kim, Radhika Dua, Jong-Yeup Kim, Eunho Yang, and Edward Choi. “Towards the Practical Utility of Federated Learning in the Medical Domain”. In: (2022). DOI: 10.48550/ARXIV.2207.03075. URL: <https://arxiv.org/abs/2207.03075> (page 2).
- [54] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. “Applied Federated Learning: Improving Google Keyboard Query Suggestions”. In: (2018). DOI: 10.48550/ARXIV.1812.02903. URL: <https://arxiv.org/abs/1812.02903> (page 2).
- [55] Honglin Yuan, Warren Morningstar, Lin Ning, and Karan Singhal. “What Do We Mean by Generalization in Federated Learning?” In: (2021). DOI: 10.48550/ARXIV.2110.14216. URL: <https://arxiv.org/abs/2110.14216> (page 17).
- [56] Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. “Adaptive Methods for Nonconvex Optimization”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: <https://proceedings.neurips.cc/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf> (pages 18, 30, 62).
- [57] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. “Dive into Deep Learning”. In: (2021). DOI: 10.48550/ARXIV.2106.11342. URL: <https://arxiv.org/abs/2106.11342> (page 60).

-
- [58] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. “iDLG: Improved Deep Leakage from Gradients”. In: (2020). DOI: 10.48550/ARXIV.2001.02610. URL: <https://arxiv.org/abs/2001.02610> (page 15).
- [59] Ligeng Zhu, Zhijian Liu, and Song Han. “Deep leakage from gradients”. In: *Advances in neural information processing systems* 32 (2019) (page 15).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

Faculté des sciences

Place des sciences, 2 bte L6.06.01, 1348 Louvain-La-Neuve, Belgique | www.uclouvain.be/sc