Alma Mater Studiorum - Università di Bologna

Dottorato di Ricerca in
COMPUTER SCIENCE AND ENGINEERING
Ciclo XXXV

Settore concorsuale:
09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore scientifico disciplinare:
ING-INF/05 SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Concepts and Methods for Efficient Decentralized Learning in Federated Settings

Presentata da: Alessio Mora

Coordinatore Dottorato:
Prof.ssa Ilaria Bartolini

Supervisore:
Prof. Paolo Bellavista

Esame finale anno 2023

Abstract

Deep Neural Networks (DNNs) have revolutionized a wide range of applications beyond traditional machine learning and artificial intelligence fields, e.g., computer vision, healthcare, natural language processing and others. At the same time, edge devices have become central in our society, generating an unprecedented amount of data which could be used to train data-hungry models such as DNNs. However, the potentially sensitive or confidential nature of gathered data poses privacy concerns when storing and processing them in centralized locations. To this purpose, decentralized learning decouples model training from the need of directly accessing raw data, by alternating on-device training and periodic communications. The ability of distilling knowledge from decentralized data, however, comes at the cost of facing more challenging learning settings, such as coping with heterogeneous hardware and network connectivity, statistical diversity of data, and ensuring verifiable privacy guarantees. This Thesis proposes an extensive overview of decentralized learning literature, including a novel taxonomy and a detailed description of the most relevant system-level contributions in the related literature for privacy, communication efficiency, data and system heterogeneity, and poisoning defense. Next, this Thesis presents the design of an original solution to tackle communication efficiency and system heterogeneity, and empirically evaluates it on federated settings. For communication efficiency, an original method, specifically designed for Convolutional Neural Networks, is also described and evaluated against the state-of-the-art. Furthermore, this Thesis provides an in-depth review of recently proposed methods to tackle the performance degradation introduced by data heterogeneity, followed by empirical evaluations on challenging data distributions, highlighting strengths and possible weaknesses of the considered solutions. Finally, this Thesis presents a novel perspective on the usage of Knowledge Distillation as a mean for optimizing decentralized learning systems in settings characterized by data heterogeneity or system heterogeneity. Our vision on relevant future research directions close the manuscript.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Paolo Bellavista, for his invaluable guidance, support, and encouragement throughout my PhD program.

I am also deeply grateful to Professor Irina Rish for providing me with the opportunity to spend my visiting period at Mila and for making me feel at home.

Furthermore, I am thankful to my colleagues at the Lab, who shared this journey with me.

I would also like to express my heartfelt gratitude to my friends and family. Their encouragement and unwavering affection have been my safe haven during the challenging times.

My immeasurable gratitude goes to Anna, who always supported me with love and patience especially during the most difficult moments.

My admiration goes to my mother, whose strength has been a constant inspiration for me.

Finally, my thoughts go to my father, who was my first and greatest supporter. I know he would have been proud and euphoric to read this dissertation, and I dedicate this achievement to his memory.

Contents

List of Figures

List of Tables

1 Introduction

The unprecedented amount of data being generated at the edge of the Internet network — the number of IoT devices has surpassed the global population and is expected to reach an astounding 80 billion by 2025 [175] — represents the ideal ingredient for training accurate Machine Learning (ML). In particular, Deep Learning (DL) models [99] allow to enhance and support a wide range of more intelligent applications, services, and infrastructures, such as powering recommender systems [215], developing data-driven machine health monitoring [219], enabling new ways for clinical diagnoses [134], or driving the design of new generation mobile networks [209]. However, the potentially sensitive or confidential nature of gathered data poses privacy concerns when managing, storing, and processing those data in centralized locations. At the same time, the capacity of the network infrastructure risks to be saturated by such continuous data collection, such as from distributed sources at the network edge to centralized cloud resources.

To this purpose, decentralized learning has recently gained momentum exactly to decouple model training from the need of directly accessing raw data, by becoming a promising alternative solution to the more traditional cloud-based ML. In fact, decentralized learning leaves the training data distributed and supports the learning of joint models via local computation and periodic communication: data no longer need to leave the data owner. For example, data remain on the premises of organizations or institutions that may want to collaborate, but without sharing their private data. Other significant use cases embrace intelligent applications for end-users of smartphones or IoT devices, where the private preferences or habits sensed through user-device interaction do not leave the source devices.

The literature includes several differently designed approaches to enable decentralized learning. The common key idea is to be able to just transmit ephemeral locally-computed updates (e.g., model parameters or gradients) and/or meta-level information (e.g., activations in neural-networks), which are meaningful only with respect to the current global model and typically bring significantly lower informative content compared to the raw data (data processing inequality). This design paves the way to upgrading the user's privacy so to meet the rising legislative requirements about it (e.g., the California Consumer Privacy Act [27] and the European General Data Protection Regulation (GDPR) [49]). Similarly, in the case of federated deployment environments participated by different institutions, the use of decentralized learning techniques can ensure privacy guarantees, especially in sensitive domains such as healthcare where data sharing is impeded by regulation (e.g., the Health Insurance Portability and Accountability Act - HIPAA [69]).

Besides the above privacy concerns, decentralized learning techniques are strongly motivated from the infrastructural perspective. The huge amount of raw data coming from the edge of the network and headed to datacenters risks to overwhelm the network backbone, hence a part of these data should, instead, be consumed locally, as suggested in [37]. Note that, even with decentralized learning, the periodic exchange of uncompressed updates in place of the upload of all the raw data may not necessarily reduce the total communication cost needed to train a model in a satisfying way [119]. Lastly, but not less important, in some circumstances collaborative learning at the edge can have a reduced carbon footprint with respect to cloud-based ML, and can be a greener technology than data center GPUs [151].

Although collaborative learning systems can open up new opportunities for utilizing a wider range of data in data science pipelines, their implementation can be challenging due to the peculiarities of federated environments, such as the variations in data statistics among learners, the limited and diverse hardware of participating devices, the possibility of still leaking sensitive information during the process, and the complexity of optimizing the process in a distributed environment [14].

The first part of this Thesis provides a comprehensive overview of decentralized learning systems. Through the lens of an original taxonomy from [14], this Thesis highlights the principal characteristics of collaborative learning solutions, presenting the concepts, the main baselines and the challenges faced in the very recent literature. In the second part of the Thesis, we focus on solutions for specific issues. We firstly present and evaluate our original methodologies and algorithmic solutions to address communication efficiency and system heterogeneity. Then, we propose an in-depth review and classification of the state-of-the-art contributions to reduce the impact of data heterogeneity, with empirical evaluations and discussion of the most promising approaches. This Thesis also provides a novel review of the literature under an original perspective: employing Knowledge Distillation as a mean to optimize collaborative learning systems, with particular emphasis on system and data heterogeneity.

1.1 Structure of the Thesis

After briefly introducing the motivations of this Thesis, this Chapter summarizes the candidate's contributions beyond the state of the art. The remainder of this document is based on published or in review papers, which are listed in the last section of this Chapter.

Chapter 2 provides background information on the topic, the description of federated settings, and an original taxonomy to guide the reader through a literature review of the principal baseline algorithms.

Chapter 3 introduces the major issues in the field and the principal research directions emerged in the last years, with a review of significant contributions for each issue.

Chapters 4 presents and empirically evaluates a novel solution for addressing system heterogeneity and communication efficiency in cross-device settings, which takes inspiration from the Dropout technique originally proposed in regular Deep Learning contexts.

Chapter 5 presents and empirically evaluates a novel solution explicitly designed for Convolutional Neural Networks (CNNs) to further enhance the communication efficiency of a state-of-the-art method, namely Sparse Ternary Compression. The code used in the experiments is also provided.

Chapter 6 provides an in-depth review and classification of state-of-the-art methods, which are inspired by the Knowledge Distillation framework from non-federated settings, to address specific issues in federated environments.

Chapter 7 explores and discusses existing solutions to address data heterogeneity, including experimental results and a comparison of different methods, shedding light on their strengths and possible weaknesses. The code used in the experiments is also provided.

Chapter 8 concludes this Thesis with a summary and details the most promising future research directions from a general perspective, and from the specific work carried out by the author.

1.2 Contributions beyond the state of the art

Conceptual Contributions:

- In [14] we provided a detailed and up-to-date overview of the major contributions available in decentralized learning literature. In particular, such survey paper originally provides the reader audience with a clear presentation of the peculiarities of federated settings, with a novel taxonomy of decentralized learning approaches, and with a detailed description of the most relevant and specific system-level contributions of the surveyed solutions for privacy, communication efficiency, data heterogeneity, device heterogeneity, and poisoning defense. Chapter 2 and Chapter 3 are partially based on this work.

- In [137] and in Chapter 6, we provide a detailed review of state-of-the-art contributions in the Federated Learning literature which employ Knowledge Distillation as a mean for optimizing the learning process, which represents a novel perspective, and we are not aware of similar contributions in literature.

- In Chapter 7, we review state-of-the-art algorithms to tackle the degradation in performance introduced by data heterogeneity, firstly classifying them in two macro categories, and then clustering them in subcategories according to their insights, rationale, and possible inspirations.

Technical and Experimental Contributions:

- In [13] we claimed that Federated Dropout can efficiently cope with device heterogeneity by exploiting a server that broadcasts custom and differently-sized submodels, selected from a discrete set of possible sub-models, to match the computation capability constraints of FL clients. In addition, we further reduce the up-link communication cost by applying per-layer or traditional Sparse Ternary Compression (STC) to sub-model updates. We demonstrate the effectiveness of our solution by reporting results for a well-known neural networks used for classification tasks considering the Federated EMNIST dataset. Chapter 5 is partially based on the work in [13].

- In [136], we proposed an original variant of Sparse Ternary Compression that is specifically designed and implemented for convolutional layers. Our variant is originally based on the experimental evidence that a pattern exists in the distribution of client updates, namely, the difference between the received global model and the locally trained model. In particular, we have experimentally found that the largest (in absolute value) updates for convolutional layers tend to form clusters in a kernel-wise fashion. Therefore, our primary novel idea is to a-priori restrict the elements of STC updates to lay on such a structured pattern, thus allowing us to further reduce the STC communication cost. We have designed, implemented, and evaluated our novel technique, called Structured Sparse Ternary Compression (SSTC). Reported experimental results show that SSTC shrinks compressed updates by a factor of x3 with respect to traditional STC and with a reduction up to x104 with respect to uncompressed ones, at the expense of negligible degradation of the global model accuracy. Chapter 4 is partially based on the work in [136].

- In [135], we originally provide a review of the most relevant related solutions in the literature to alleviate the harmfulness of non-identically and independently distributed (IID) data, highlighting the intuition behind these alternative strategies as well as their possible drawbacks. Furthermore, we propose an empirical comparison among a subset of such state-of-the-art solutions under different levels of data heterogeneity running them in the same operating conditions. We end up identifying the most promising approaches considering both empirical performances and defining characteristics (e.g., assumptions the strategy possibly make).

- In Chapter 7, we evaluate a set of recent state-of-the-art algorithms in challenging settings of simulated data heterogeneity.

- We provide the reference to TensorFlow code we used for experiments of Chapter 7, which contains the implementations of several state-of-the-art baselines, and can be relevant for the research community, given the fact that the vast majority of available code does not use TensorFlow (e.g., based on Pytorch instead). The code is fully reproducible thanks to the use of seeds, and the complete specification of client dataset composition. We also provide the code to generate client data shard with a distribution-based label skew method (explained in Chapter 3).

1.3 List of Publications

The following list contains all contributions made by the author to the field of decentralized learning. Some of the materials presented in this dissertation have been previously published as journal articles, presented at scientific conferences or still being in the process of peer review.

1. Bellavista Paolo, and Alessio Mora. "Edge Cloud as an Enabler for Distributed AI in Industrial IoT Applications: the Experience of the IoTwins Project." AI&IoT@ AI* IA. 2019.

2. Bellavista Paolo, Luca Foschini, and Alessio Mora. "Decentralised learning in federated deployment environments: A system-level survey." ACM Computing Surveys (CSUR) 54.1 (2021): 1-38. DOI: https://doi.org/10.1145/3429252

3. Bellavista Paolo, Luca Foschini, and Alessio Mora. "Communication-Efficient Heterogeneous Federated Dropout in Cross-device Settings". GLOBECOM 2021 - 2021 IEEE Global Communications Conference, 2021, pp. 1-6.
DOI: 10.1109/GLOBECOM46510.2021.9685710.

4. Alessio Mora, Luca Foschini and Paolo Bellavista. "Structured Sparse Ternary Compression for Convolutional Layers in Federated Learning." 2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring), 2022, pp. 1-5,
DOI: 10.1109/VTC2022-Spring54318.2022.9860833.

5. Alessio Mora, Irene Tenison, Paolo Bellavista, Irina Rish. "Knowledge Distillation for Federated Learning: a Practical Guide." arXiv preprint arXiv:2211.04742 (2022).

6. Alessio Mora, Davide Fantini, Paolo Bellavista. "Federated Learning Algorithms with Heterogeneous Data Distributions: An Empirical Evaluation." The Seventh ACM/IEEE Symposium on Edge Computing 2022. DOI: 10.1109/SEC54971.2022.00049

7. Alessio Mora, Irene Tenison, Paolo Bellavista, Irina Rish. "Knowledge Distillation in Federated Learning." Under review. Submitted to ACM Computing Surveys (CSUR).

2 Decentralized Learning in Federated Settings

This Chapter firstly presents the motivations that led decentralized learning techniques to rise as an alternative to cloud-based ML and illustrates some real-life examples of its application. Next, it provides a characterization of possible federated settings as well as an original taxonomy to highlight the primary design choices when building a collaborative learning system. The Chapter also details relevant baseline algorithms, such as Federated Averaging, that has emerged in the recent literature.

2.1 The Rising of Decentralized Learning

The public opinion is becoming increasingly sensitive to individual privacy rights, especially after the notorious Facebook-Cambridge Analitica scandal [192] has made no longer ignorable the Orwellian levels of data held by such companies about us and has exposed the weakness (or even the non-existence) of privacy regulation and data protection. Anyway, even without thinking to striking episodes such the above cited one, individuals' privacy is threatened whenever personal raw data are disclosed. For example, elementary data anonymization (i.e., removing all explicit identifiers such as name, address, and phone number) has demonstrated to be almost ineffective in protecting privacy, since combinations of simple non-unique attributes often allow to re-identify individuals by matching "anonymized" records with non-anonymized ones in a different public dataset (e.g., [139]).

The actual legislative vacuum about data harvesting, data holding, and data processing has been — and still is — the subject of regulation efforts around the world. About that, it is worth mentioning the CCPA and the GDPR, respectively from California and from

European Union, that both leverage the principles of *purpose specification* and *data minimization*. In concrete terms, for example, the GDPR's Article 5 states that personal data shall be "collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes" and "kept in a form which permits identification of data subjects for no longer than is necessary for the purposes for which the personal data are processed". Such guidelines are often incompatible with more traditional cloud-based ML solutions, where potential privacy-sensitive raw data flow towards datacenters to train ML/DL models. In particular, (i) companies harvesting data tend to keep them forever and users cannot delete them[1], hence same data can be used several times for different learning purposes (for extracting different kinds of insights); (ii) users from whom the data were collected are unaware of the associated learning objectives; (iii) models trained on collective data typically remain property of the companies that built them; and (iv) users disclose their raw data, in a more or less informed way, to infer centralized models, such as for training.

It could seem that an inevitable dichotomy between the protection of individual's privacy and the distillation of useful knowledge from a population exists (i.e., not disclosing private data to preserve privacy, by merely performing local learning, versus sharing private raw data to produce more accurate models at the cost of exposing data owners to privacy violation risks). On the opposite, decentralized learning tries to alleviate the privacy concerns of traditional cloud-centric training by design and is data-minimization-prone. In fact, (i) companies do not need anymore to collect possible privacy-sensitive raw data to build ML/DL models; (ii) users could likewise be unaware of the learning objective for which their data are used, but data processing happens locally, hence facilitating the shift to full transparency; (iii) models (or fractions of models, i.e., portions of their parameters) reside locally at the user's device or inside the organization's premises (or in very proximity of it). This could be seen as a first step to give back to the community the knowledge acquired from joint contributions[2]; (iv) users do not need to upload their raw data to query centralized models, in fact on-device inference is typically enabled if the entire model is replicated locally — if only a portion of the model parameters is lo-

[1] At least until the time this thesis has been written.

[2] However, it is worth noting that restricting or preventing access to model's parameters, even if the model itself is locally available, makes it harder for an attacker to undermine it, e.g., via backdooring. Therefore, companies or organizations that adopt Decentralized Learning techniques may be anyway motivated to hamper model inspection.

cally held instead, distributed inference is performed by just communicating meta-level information in place of raw data.

In addition, shifting model training from the cloud towards the network edge recalls a trend that was already in act with the rising of mobile edge computing during the last decade. Besides the urge of privacy guarantee, several aspects are similar and seem to overlap. A primary one is the need to relief the burden on the backbone of the network infrastructure, which risks to collapse under the tsunami of data if not partially consumed locally or in proximity of the associated sources. Intuitively, actively involving the ecosystem of edge devices in the learning process and exchanging model updates in a communication-efficient way (e.g., employing stream compression) in place of centralizing raw data can substantially reduce network traffic while leading to limited degradation (or in some cases to no degradation) of model accuracy. Secondly, the low-latency requirements of real-time applications often cannot be met by only leveraging the cloud (for instance when monitoring a shared industrial workspace, during human robot collaboration, to enforce policies for worker protection [163]). Enabling on-device inference of the learned or in-learning models, which naturally comes with most decentralized learning approaches as we will discuss in the continuation of the thesis, benefits such delicate aspect. Let us finally note that decentralized training, with its potential reduction of ML-related energy consumption because of reduced network traffic and decreased transmission distance, also contributes to the overall sustainability of the approach: it is considered as one of the key enabling technologies towards green networking via distributed and federated datacenters [151].

Decentralized learning finds natural applications in smart apps for mobile devices which learn by user interaction, and where low-latency responses are required. In this context, gathering user-labeled or automatically annotated data points for feeding supervised learning algorithms is a common practice. Related examples include on-device intelligent keyboards that power content suggestions [202], or that predict the most suitable next words [61] or the most fitting Emojis [153] given the chat history; or again vocabularies that evolve to follow the ongoing trending expressions by learning out-of-vocabulary words [31], and all of this without exporting sensitive text to servers. Other examples deal with human activity recognition (e.g., [173]) and keyword spotting for voice assistants in smart homes (e.g., [102]) .

Decentralized learning has been used also to conjugate user privacy and prediction ability of the infrastructure in the 5G multi-access edge computing architecture [43, 92, 127], for example for proactive content caching [207] or for optimal allocation of virtual machine replicas copies [50], and it is considered a key enabling tool for next generation wireless networks [141] as well, e.g., for spectrum management.

Confirming its versatility, decentralized learning has been also applied to network traffic classification, anomaly detection, and VPN traffic recognition tasks, while preserving appropriate privacy levels [220] [12]. Similar considerations apply to vision-based safety monitoring systems in smart cities [122].

In the relevant healthcare domain, the popularity of decentralized training approaches has been also pushed by the need to enable collaboration among healthcare institutions. In fact, the disclosure of patients' raw data is often impeded or limited by regulations such as the HIPAA Privacy Rule, or the patient herself might not want her clinical data to be released to other entities, or again the institutions might not want to sell out their valuable datasets. Therefore, plain old centralized training results to be not feasible for predictive clinical models in many cases. Furthermore, manual labeling of data is often very time-consuming in medical contexts and typically requires qualified personnel. Datasets held by single institutions tend to be small and may lack in diversity [145], and this is exacerbated when considering rare diseases. Hence, from the perspective of isolated local learning, sample scarcity may lead to models with poor predictive ability, especially when considering deep learning models that notoriously need abundant data points to reach high fidelity. As practical use cases in smart healthcare, we report the training of a detector for abnormal retinal fundus and a classifier for common chest radiography observations (from visual datasets) [150]. Other clinical learning tasks include prediction of prolonged length of stay and in-hospital mortality [147], prediction of hospitalizations for cardiac events [22], or gaining insights about brain diseases [158].

2.2 Fundamentals. Taxonomy and Baselines for Decentralized Learning

This Section gives some concise background to make highly accessible the following presentation of diverse decentralized learning solutions, by defining the targeted deployment settings and the modular building blocks that are emerging in the related literature. These building blocks are at the cornerstones of our original taxonomy, which we will introduce in this Section and use in the remainder of the thesis to better highlight the features, the pros, and the cons of the surveyed contributions. We also present the most interesting baseline solutions to enable decentralized learning.

2.2.1 Cross-Silo and Cross-Device Federated Settings

Here we provide an informal and qualitative characterization of the two most common settings for decentralized learning, by highlighting their specific elements with respect to traditional distributed settings [39]. As anticipated in the previous sections, decentralized learning techniques are strongly motivated when data sharing is impeded by law or by privacy concerns, hence they apply to several real-world contexts. For the sake of simplicity, let us consider two extreme scenarios: (i) the federation of entities participating in collaborative learning tasks consists of compute nodes from different organizations or companies (e.g., hospitals, banks) — that typically store their private data in on-premise silos —; (ii) the federation comprises a massive amount of edge devices (such as smartphones, IoT devices, or IIoT devices). Such primary distinction leads to the identification of two very general settings, which are respectively referred to as Cross-silo federated settings and Cross-device federated settings [89]. The comparison among settings is also summarized in Table 2.2.1.

Those two federated scenarios are substantially different from more traditional distributed settings, where raw data are centralized in datacenters to perform learning. In fact, in cloud-centric training, the participants of the learning task are compute nodes (generally up to 1000) interconnected through very fast networks, making the computation cost the major bottleneck. Data can be balanced across compute nodes; moreover, they can be partitioned and re-partitioned according to the need. Importantly, any participant can access any part of the dataset. Worker machines are reliable and low rate of failure or drop out (i.e., abandoning the learning task without notice) are expected.

The Cross-silo federated setting refers to a scenario in which the entities involved in the learning process are limited in number (up to 100 participants), and typically they are trusted and reliable. In addition, they are likely to participate in the entire training task. Data can be unbalanced, but in general not as much as in Cross-device settings. No

Table 2.1: Defining characteristics of data-center distributed learning, cross-silo federated learning and cross-device federated learning.

Setting	Data-Center Distributed Training	Cross-silo Federated Learning	Cross-device Federated Learning
Description	Clients are compute nodes in a datacenter environment or in a single cluster.	Clients are organizations, companies or geo-distributed datacenters.	Clients are edge devices, e.g. IoT devices, IIoT devices.
Data distribution	Data on clients are supposed to be identically and independently distributed (IID); they can be arbitrarily partitioned, shuffled and balanced across clients. Any client can read any part of the dataset.	Data is stored locally and remains distributed. Each client cannot access the data of other clients. Data is supposed to be not independently or identically distributed (non-IID).	
Data availability	Clients are supposed to be available almost always.		Only a subset of the total clients may be available at a certain point in time (e.g. clients may be not eligible for a learning rounds if they have low battery or scarce network connection).
Client reliability	Low rate of failure is expected.		Clients can drop out without notice.
Client scale	Up to 1000 clients.	Up to 100 clients.	Up to 10^{10} clients.
Principal bottlenecks	Very fast network are supposed to be available; computation dominates communication bottlenecks.	No assumptions about communication or computation bottlenecks are made a priori.	Both communication and computation represent bottlenecks. Also, the federation is supposed to be heterogeneous in the hardware capabilities on clients.
Example		Health institutions such as hospitals.	A federation of smartphones.

assumptions about communication or computation bottlenecks are made a priori. Furthermore, while training data are assumed to be independently and identically distributed (IID) in typical datacenter settings, such assumption does not hold for federated settings (neither for Cross-silo nor for Cross-device): the training data on a given device or on a given machine are likely not to be representative of the full population distribution.

In the Cross-device federated settings, participants are very numerous instead (up to 10^{10}), data are massively distributed and unbalanced (e.g., the number of training examples held by participants can differ by one or two orders of magnitude) [96]. Learners are highly unreliable; failure and drop out must be addressed, and each client is likely not to take part in the entire training process (actually they may contribute only once per task). Furthermore, since edge devices have limited bandwidth, communication efficient solutions are preferable in Cross-device setting; the federation may comprise computationally constrained devices as well, making more delicate the computation/communication trade-off. Another peculiarity is that participants may be malicious in this scenario, e.g. trying to infer sensitive information about other learners or voluntarily hampering the global learning.

For the sake of clarity, we use this characterization[3] to readily approximate the setting to which the referenced works in Chapter 3 refer — we will show that the targeted federated setting relevantly influences the design choices of a solution. We indeed use such characterization of the setting as a primary dimension of our taxonomy.

2.2.2 A Taxonomy for Decentralized Learning Systems

To favor the readability of the remainder of this Section and of the Thesis, we propose a taxonomy for decentralized learning systems that highlights the main alternative options in designing such frameworks.

Data processing: Data-sequential vs Data-parallel

The common thread when designing decentralized learning algorithm is leveraging data-parallel variants of iterative optimization algorithms that are inherently sequential, e.g.

[3]We use the terminology found in [89]. However, the existence of a central orchestrator (i.e., an entity orchestrating the collaborative training) in federated settings, either Cross-silo or Cross-device, is further supposed in [89]. To embrace all the decentralized learning work from the literature, we relax this last trait in our terminology usage in this paper.

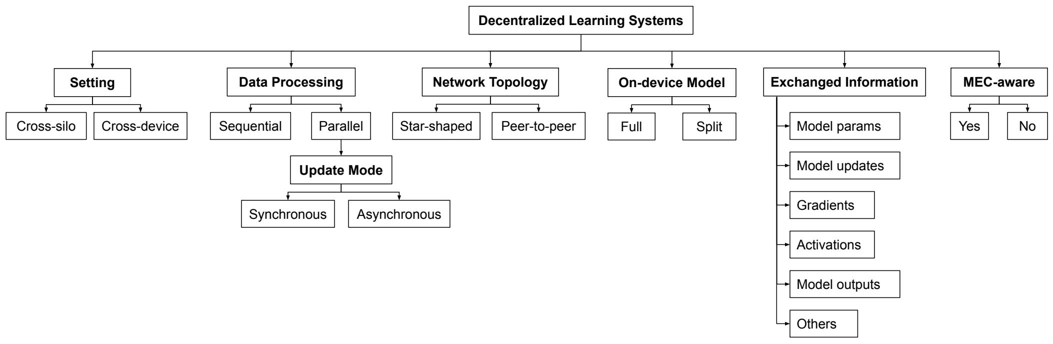


Figure 2.1: A taxonomy for decentralized learning systems.

Stochastic Gradient Descent (SGD) and its optimizations. Typically, the federation of learners collaborates to minimize a global objective function, that is unknown to the participants since no single node has direct access to all the data. The global objective can be thought as a linear combination of the local empirical losses, available locally to the participants [96].

We further divide data-parallel approaches into systems that leverage *synchronous* or *asynchronous update mode*. In fact, as traditional distributed training algorithms, also data-parallel decentralized learning approaches can exploit asynchronous updates to optimize on speed by using potentially stale parameters for local training or wait for local computation of the slowest participant to synchronously aggregate updates without risking to use outdated parameters. With synchronous update mode, it is usual to talk about rounds of communication, i.e., all the triggered participants retrieve the global model state, produce their locally computed updates and communicate such updates, from which the new generation model will be derived. Communication efficient algorithms have their principal goal in minimizing the rounds of communication. Relaxing the synchronicity can instead spread the communications over time, particularly helpful when handling a large number of learners. However, examples of data-sequential systems exist, i.e., systems in which each participant uses as starting model state the result of the computation of another participant, and thus produces as output the input model state for the next participant. Anyway, let us note that these solutions are usually limited to the Cross-silo setting.

Network Topology: Star-shaped vs Peer-to-peer

The coordination among learners can be facilitated by a star-shaped network topology that leverages a central entity to distribute the current state of the global model at the beginning of each local iteration, and maintain the state updated during the training task. Participants can directly exchange their locally computed updates as well, in a peer-to-peer fashion, hence not requiring any infrastructure at the price of increased coordination complexity. In literature, decentralized learning frameworks that exploit peer-to-peer networks of participants are often referred to as fully decentralized, i.e., decentralized in both data and coordination.

On-device Model: Full Model vs Split Model

Besides the full local replication of the (current) global model during the training process, it can be possible to have participants that are only responsible for a fixed subset of model parameters (in this case, typically, the parameters belonging to n shallower layers in a deep neural network, i.e. split models). The full replica of the global model enables on-device inference by design, while in the case of split model, without retrieving the entire model at the end of the training, distributed inference is required. Note that, anyway, the primary privacy concerns have been bypassed by having feature extraction locally[4].

Exchanged knowledge: Model Parameters/Updates, Gradients, Activations and Others

We also emphasize that the degrees of freedom in designing decentralized learning frameworks also involve the kind of exchanged information during the distributed learning. Supposing gradient descent based methods for optimization, the usual practice is to have participants exchanging gradients or model updates, with the latter option valuable in case of participant-specific local solver. In star-shaped topology, a common practice is to have participants downloading the current model parameters and communicating back to the aggregator either the gradients, the locally updated model parameters typically generated through SGD iteration(s), or the locally computed updates for the global model (i.e., the difference among the received global parameters and the locally computed ones).

[4]It is important to remind that information leakage is still possible. This will be faced in Section 3.2.

While the most common strategy is to have the server broadcast the global model parameters and collects model updates[5], there are examples of star-shaped frameworks where the communication in both the directions only involves gradient information (e.g., [180], [15]) as well, i.e., the server aggregates gradients and the back-propagation is performed on-device. We underline that the exchanged information may be not limited to gradients and model parameters, in fact other kinds of parameters may be transmitted for diverse optimization purposes. For instance, the exchange of moment estimates to implement an ADAM[95]-inspired optimization algorithm [133], or also of information for gradient correction terms [109], and of control variates [91] to tackle non-IIDness, or of other local estimations to meet given budget resources [190]. Or again, in presence of split models (e.g., in Split Learning), besides model parameters and gradients, also activations (and labels) have to be communicated by design. Also, as presented more in detail in Chapter 6, information based on model outputs (or more in general on model responses) can be exchanged in place of model parameters, providing advantages which span from reducing the communication cost of the process to enabling model heterogeneity among clients, while possibly introducing other trade-offs.

MEC-awareness

It is also worth mentioning that, considering the MEC architecture and therefore the existence of a middle layer of edge servers between the edge devices and the cloud, two levels of topology organization can be identified. On the one hand, decentralized learning systems may leverage edge servers as intermediate aggregators for updates produced by the edge devices in their locality (i.e., matching a star-shaped topology) and then edge servers may directly exchange intermediate-level updates among them in a peer-to-peer fashion, to collaboratively build the global model. On the other hand, the cloud may be involved as "master aggregator" collecting intermediate aggregations from the federation of edge servers (the latter solution is referred as *hierarchical*). An in-depth discussion about edge-cloud continuum roles in edge intelligence can be found in [226].

[5]Sending back updates instead of model parameters have at least three major advantages: (i) they can be interpreted as gradients for a server-side optimizer of choice (this will be presented more in depth in Chapter 7), (ii) they are more suitable for compression, (iii) they leak less information.

2.2.3 Baselines for Decentralized Learning Systems

In this subsection, we propose some baseline frameworks to enable decentralized learning. We introduce the most significant baselines for star-shaped systems, followed by instances of fully decentralized (server-less) alternatives, i.e. peer-to-peer.

Star-shaped Baselines

Federated Averaging (FedAvg) is a widely accepted heuristic algorithm used as baseline for star-shaped Federated Learning (FL), given its simplicity and its empirical effectiveness [128] also in non-convex setting. Its skeleton is presented in Algorithm 1. The learning process proceeds in synchronous rounds of communication; the (full) current global model is broadcast at the beginning of the round to the (selected) participants, that use their private dataset to produce an update (e.g., gradients or model weights) for the received model, and upload such contributions. The aggregator, i.e. a sort of parameter server, collects and aggregates (e.g., by averaging) the updates from participants and computes the new-generation global model. The process typically ends when a certain accuracy for the global model is reached, or when a certain number of rounds has been executed. SGD is typically chosen as local solver. Four hyperparameters have to be tuned in FedAvg; C controls the fraction of participants to be selected in a certain round t (with $C = 0.0$ indicating only one participant involved per round, and $C = 1.0$ meaning the totality of participants), E defines the number of local epochs to be performed in each round, B denotes the minibatch size, and η the local learning rate. It is worth noting that the contributions in the aggregation are weighed accordingly to the number of local data points held by each participant.

When the full local dataset is treated as a single minibatch (i.e., $B = \infty$), and the local iterations at each participant are limited to one epoch (i.e., $E = 1$), FedAvg is also known as FedSGD. An equivalent variant of FedSGD can be formulated by uploading gradients in place of model parameters.

An accurate convergence analysis, in strongly convex and smooth problems, of FedAvg in presence of data heterogeneity and partial device participation — peculiar of cross-device settings — can be found in [111]. The authors theoretically showed that, in such circumstances, model convergence is slowed down with respect to the ideal case of IID-ness and full participation. They also pointed out that a decaying local learning rate is

Algorithm 1: FedAvg algorithm
The K participants are indexed by k, $\mathcal{D}_k$ is the local dataset at participant k, $n_k = |\mathcal{D}_k|$ and $n = \sum_{k=1}^{K} n_k$, B is the local minibatch size, E represents the number of local epochs, η is the learning rate. Note the common initialization of model parameters w_0.

```
1  Server executes:
2      initialize $w_0$
3      for each round $t = 1, 2, 3, ..$
4          $m \leftarrow max(C \times K, 1)$
5          $S_t \leftarrow$ (random set of $m$ clients)
6          for each client $k \in S_t$ in parallel
7              $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
8          $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
9  ClientUpdate$(k, w)$
10     $\mathcal{B} \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
11     for each local epoch $e$ from 1 to $E$
12         for batch $b \in \mathcal{B}$
13             $w \leftarrow w - \eta \nabla \ell(w; b)$
14     return w to server
```

fundamental for the convergence of FedAvg under non-IIDness: gradually diminishing the learning rate can neutralize biased local updates. Considering FL-suitable participant sampling and related averaging schemes, the authors of [111] establish a convergence rate of $\mathcal{O}(\frac{1}{T})$, where T represents the total number of SGD iterations performed by every participant.

FedAvg is considered a communication efficient algorithm mainly thanks to two aspects: (i) it selects a (random) subset of participants per round (i.e., if only a portion of participants is selected, the per-round communication cost is reduced with respect to full participation); (ii) it allows for additional iterations of local solver (i.e., SGD) to reduce the total number of synchronizations needed for model convergence – it has been empirically showed that FedAvg significantly reduces the total communication rounds (under the same C-fraction of per-round selected clients) with respect to FedSGD, while reaching the same (or higher) model accuracy [128]. A plethora of works in literature propose improvements for FedAvg (as presented in the next Chapters).

Star-shaped alternatives to FedAvg exist in literature as well. The major instances are represented by solutions that exploit adaptations of regular Knowledge Distillation (KD) – such a class of algorithms will be treated in depth in Chapter 6. For the sake of this system-level analysis, it is worth mentioning that such KD-based solutions do not need to exchange model parameters, but they transfer knowledge based on the client model outputs. For example, Federated Distillation (FD), is presented in [83], and it is explicitly designed to be extremely communication efficient; it is inspired by an online version of knowledge distillation, namely co-distillation [71], [8]. In a nutshell, each device (the student) stores its model outputs, i.e. a set of logit values normalized via softmax function, from which it derives per-label mean logit vectors, and periodically uploads such local-average logit vectors to the aggregator. The server produces the per-label global-average logit vector by averaging the contributions of all the participants in that round, and broadcasts such aggregation to the federation; each device locally regularizes training with a distillation term derived from the received global logit vector (i.e., their local loss becomes a linear combination of two terms). It is straightforward to note that exchanging logit-vector (local or global averaged, whether they are upload or download parameters), in place of model parameters or gradients, reduces the per-round communication cost with respect to FedAvg: the dimension of logit-vectors depends on the number of labels, and not on the number of model parameters [6].

A differently designed method to enable collaborative training of neural networks without sharing raw private data is the so-called Split Learning (SL), also referred as SplitNN [59] to emphasize the suitability for DL architectures. This technique employs *split models* instead of *full model replication*. In fact, the training participants hold replications of the shallower layers up to a certain layer (i.e., the *cut layer*), and a central entity holds the deeper layers. Inter-layer values, i.e., activations and gradients exchange occurs between a certain participant and the central entity, instead of centralizing the raw data.

The training process as formulated in [59] is data-sequential, albeit distributed. Each participant retrieves the current state of the shallower layers of the neural network either in a peer-to-peer mode, downloading it from the last training participant, or in a centralized mode, downloading it from the central entity itself, and runs the local gradient

[6]Referring to a classification task performed via a neural network.

descent based local solver (e.g., SGD), using its private dataset[7]. The participant computes the forward propagation up to the *cut layer*, and the outputs of this layer, together with label associated to the data examples, are communicated to the central entity that concludes the forward pass on the deeper layers. The back propagation of gradients takes place in a similar fashion, flowing from the deepest layer to the cut layer, where they are sent from the central entity to the participant that has initially triggered the forward propagation (only the gradients that refers to the *cut layer*). Then, the process repeats with a different participant, collectively learning a joint model without sharing private raw data. In [170] the position of the *cut layer* is empirically discussed.

Authors of [59] also proposed a variant of the SplitNN algorithm, namely U-shaped Split Learning, in which the labels related to the locally available training examples are not centralized but remains private at the participant side.

A data-parallel variant of SplitNN is proposed in [182], namely SplitFed learning (SFL), to combine the advantages of FL and SL, that are respectively the parallel processing among distributed learners and the model partitioning among participants and central entity.

Although splitNN has demonstrated to reduce computation burden and bandwidth utilization with respect to baseline FedAvg [170] in presence of "big" models and high number of clients, star-shaped FL and fully decentralized FL allow on-device inference of the model by design, while this is not true for splitNN that requires a distributed inference unless the complete trained model is provided to the participants.

Peer-to-peer baselines

In star-shaped FL, the coordination server orchestrates the communication rounds; it iteratively broadcasts the current model state to the participants and gathers the locally computed updates to produce the next-generation model by aggregation. Although leveraging a client-server architecture permits to ignore topology-related issues, FL presents two downsides: (i) the central entity can be seen as a single point of failure; (ii) the central entity may represent a bottleneck considering a significant number of training participants (as demonstrated in [114] though not explicitly targeting federated settings). Fur-

[7]Regardless of the strategy to retrieve the current state of the participant-side model, either peer-to-peer or centralized, in SplitNN a server exists by design; this is why we consider it as star-shaped.

thermore, the learners should trust such central aggregator, and, even though techniques such as multi-party computation can ensure inscrutability of updates (see Section 3.2), the participants may prefer to coordinate each others directly (as could be the case of health institutions).

In fully decentralized learning, the topology of star-shaped FL becomes a peer-to-peer topology, represented as a connected graph (generally assumed to be sparse). Such graph can be a directed graph or an undirected graph, i.e. unidirectional or bidirectional channels of communication among the nodes. The topology can be assumed to be fixed or dynamic, i.e. in which interconnections between nodes may change over time.

In each round, participants perform local computation and then communicate with (a subset of) the other nodes in the graph — note that not leveraging the server-client architecture (as well as relaxing the synchronous update mode) redefines the semantic of *rounds*. Straightforward optimization algorithms, similarly to FedAvg, employ fully decentralized variants of SGD (e.g., peers directly exchanging and merging gradients or model updates). It is also worth highlighting that, while in star-shaped FL the FedAvg algorithm has been widely accepted as baseline, in peer-to-peer (server-less) FL there is no algorithm that has distinctly emerged among others; solutions in literature, in fact, make different assumptions on the connectivity of the graph, in particular considering each node connected to all the other nodes in the network or considering only a set of nodes (i.e., the neighbours) reachable by each one, considering a fixed topology or a dynamic topology, assuming directed (e.g., [65]) or undirected graphs, and employing different strategies for model fusions.

In the continuation of this subsection, we present examples of baseline algorithms that consider fixed-topology and undirected graphs — most common assumptions. The first work, BrainTorrent [158], targets cross-silo federated settings, while the subsequently presented ones also embrace the cross-device setting [70] [84] [163].

BrainTorrent considers the graph as fully connected, from this consideration comes our labeling as cross-silo framework — it explicitly targets the collaboration of medical institutions, where it is reasonable to further suppose full connectivity besides fixed topology and undirected network graph. In a nutshell, a random participant k in the network starts the learning process by pinging all the others node requesting for model updates; the ones that have a fresher version of the model respond with their model parameters; the learner that has initiated the process, gathers the updates from the subset of participants that have responded, referred as $N_{\overline{k}}$, and aggregates them with its own local model by using this strategy: $\psi^k = \frac{n_k}{n} w^k + \sum_{i \in N_{\overline{k}}} \frac{n_i}{n} w^i$. Next, the participant k fine tunes the aggregated model ψ^k using its own private dataset, it updates the version of its model and it is ready to respond to ping request from other nodes by providing its new generation fine-tuned w_k. Then the process repeats.

Gossip-based protocol for distributed learning has been explored in the datacenter setting as alternative to the parameter-server approach (e.g., [16], [62]). Inspired from them, Gossip Learning (GL) has been proposed in [70] for Cross-device federated settings. In the baseline GL algorithm, starting from a common initialization, each node sends its local model to a randomly selected peer, which firstly merges (e.g., by averaging and weighing the average according to an age parameter associated with the freshness of the models) the received model with its current parameters, then updates the resulting model by exploiting its private dataset, and the process repeats. In a nutshell, there could be different models scattered across the network of peers, with each one of these models taking random walks (in the network) and being updated when visiting a new node. Typically, the local update is implemented through minibatch SGD algorithm. It is worth noting that due to the push only nature of the considered protocol, the merge-update-push cycles are not synchronized among participants: a node may merge its fresher model with an outdated one. The GL strategy, in [70], is not evaluated on DL architectures. Furthermore, this seminal work does not thoroughly discuss some aspects related to different kinds of heterogeneity that arise in real-world cross-device setting; in particular, the data held by peers, the neighbors reachable by each peer in the network, and the processing and communication speeds of devices are unrealistically supposed to be homogeneous. Such aspects are considered and discussed in [55], where it is claimed that gossip learning shows poor performance on restricted communication topologies and it is highlighted that GL fails to converge when communication speeds of the nodes and heterogeneity of data are correlated. Authors of [55] propose some strategies to improve GL in such realistic scenarios.

In BACombo [84], authors consider a fixed topology of neighbors for each learner, not limiting the spreading of the updates to one peer per round, and propose a neural-network specific solution. The local model held by each peer is splitted into a set of S not-overlapped segments, and each participant does not pull all the segments (i.e., the entire model) from the same peer but collects S segment from S different links in the

Algorithm 2: Consensus FedAvg algorithm
$N_{\overline{k}}$ represents the set of neighbors of the participant k, hence k excluded, $\mathcal{D}_k$ is the local dataset at participant k, B is the local minibatch size, η is the learning rate.

```
1  Participant k executes:
2      initialize $w_0^k$
3      for each round $t = 1, 2, 3, ..$
4          receive$\{w_t^i\}_{i \in N_{\overline{k}}}$
5          $\psi_t^k \leftarrow w_t^k$
6              for all devices $i \in N_{\overline{k}}$
7              $\psi_t^k \leftarrow \psi_t^k + \zeta_t \alpha_{t,i}(w_t^i - w_t^k)$
8          $w_{t+1}^k$ = ModelUpdate($\psi_t^k$)
9          send($w_{t+1}^k$) to neighbors
10 ModelUpdate($\psi_t^k$)
11     $\mathcal{B} \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
12     for batch $b \in \mathcal{B}$
13         $\psi_t^k \leftarrow \psi_t^k - \eta \nabla \ell(\psi_t^k; b)$
14     $w_t^k \leftarrow \psi_t^k$
15     return($w_t^k$)
```

network of neighbours. In this way, each peer reconstructs a model update by building a mixed model composed by such S segments that have been pulled from different peers. They extend the solution by allowing each peer to pull $S \times R$ segments in each round of communication, with R being an hyper-parameter, to be carefully tuned, that represents the number of mixed models that can be reconstructed, thus impacting the communication efficiency while accelerating the propagation of fresh model. The mixing strategy is similar to FedAvg, weighing contributions (i.e., segments) according to the cardinality of the dataset held by participants.

In [163], authors propose a consensus-based FedAvg-inspired algorithm (referred as CFA), supposing sparse connectivity. The algorithm is formalized in Algorithm 2. In each round, the participant k receives models from its neighbors and produces an aggregated model, ψ^k. Next, local iterations of mini-batch SGD are performed to produce the new-generation model, that will be sent to the neighbors, before the process repeats. The peculiarity of the algorithm stands in how the aggregated model is obtained, at round t, from the neighbor contributions, that is: $\psi_t^k = w_t^k + \zeta_t \sum_{i \in N_{\overline{k}}} \alpha_{k,i}(w_t^i - w_t^k)$, where ζ_t

is the "consensus step size" and the mixing weights $\alpha_{k,i}$ are chosen, similarly to FedAvg, as $\alpha_{k,i} = \frac{n_i}{\sum_{i \in N_k^-} n_i}$ with n_i being the cardinality of data samples at participant i.

We conclude this overview about instances of baseline algorithms for server-less federated learning by mentioning the fact that blockchain-based implementations of peer-to-peer learning frameworks have been — and are — explored in literature (e.g., [93]), though not being explored in this thesis.

3 Issues in Decentralized Learning

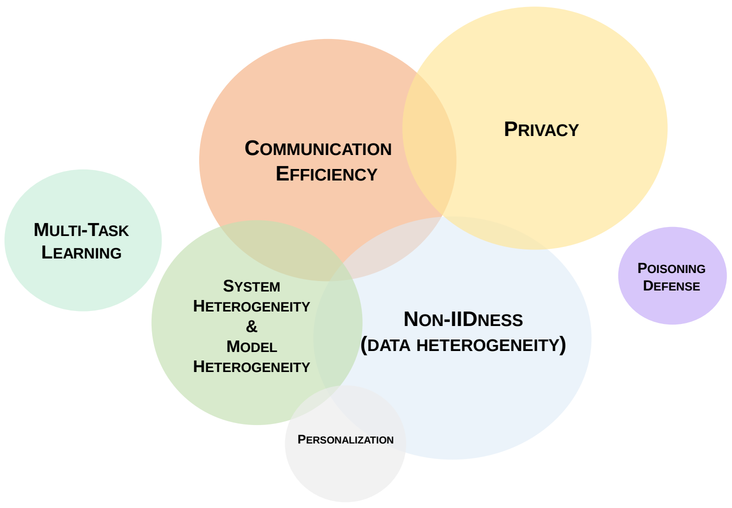


Figure 3.1: Visualization of major issues in decentralized learning.

Decentralized learning decouples by design the ability to learn a predictive ML/DL model from the direct access to raw data and meets the rising urge of ensuring privacy guarantees to the data owners while still being able to distill useful knowledge for the community. However, as already pointed out in this Thesis, diverse challenges emerge. Chief among them, privacy is not completely secured by means of just disclosing ephemeral updates (e.g., gradients, model parameters, model updates) or meta-level information, as well as the communication efficiency is of paramount importance in cross-device federated settings. Furthermore, having the raw data (massively) distributed and/or unbalanced among participants naturally implies dealing with non-IIDness. An additional factor to be addressed is the heterogeneity of devices' resources in cross-device settings (system heterogeneity), which calls for model heterogeneity. Moreover, the design of decentralized learning approaches opens up to new possibilities for attackers, since learners actively participate in the training process, e.g. forcing information leakage from other participants or trying to influence the behaviour of the system. Recently, the FL community is witnessing a growing interest to personalization, i.e., while typically the collaborative solutions are designed and evaluated on the generalization ability of the global model, in personalization the prerogative is to perform well on local data (e.g., build global models which are prone to local specialization). These are the most investigated issues in literature so far, but additional aspects and challenges are rising and taking the scene while effective solutions for the urgent aspects permit to already apply decentralized learning in real scenarios. In this section, we discuss the systems in the literature that aim at solving some of the above mentioned issues, i.e. communication efficiency, privacy, device heterogeneity, and poisoning defense. For non-IIDness of data among clients, we present the issue in detail, while leaving the analysis of the state-of-the-art related methods to Chapter 7. In the following, we do not deepen the literature related to client personalization, and we refer to [178] for this aspect. Also, discussing multi-task FL (e.g., [171]) is outside the scope of this Thesis.

Fig. 3.1 visualizes the major issues faced in FL settings. Issues in collaborative learning – and the related proposed solutions – are often correlated, i.e. a methods that aim at alleviating a specific issue may be beneficial or detrimental for another aspect. The overlapping of issues in the figure underline such a mutual influence. For example, data heterogeneity, as we will detail in the following, hampers the converge of the global model and requires more rounds to achieve consensus among nodes (e.g., more communications), with respect to an ideal IID case. As a consequence, methods that reduce the degradation introduced by data heterogeneity, simoultaneously decrease the amount of communication rounds for the convergence. In a similar vein, approaches that compresses the information exchanged during the collaborative training can have beneficial effects in terms of privacy guarantees, i.e., the exchanged payload leaks less sensitive information. However, typically such benefits come at a cost, which may be, for example, local computation or memory overhead, as we will detail later on in this Thesis.

Let us note that, in the following sub-sections, we will use the taxonomy definitions and terms introduced previously in this Thesis; where not possible or convenient, we explain in-line the specific meaning of the employed definitions/terms/symbols.

3.1 Improving Communication Efficiency

The communication efficiency in decentralized learning can be addressed from different perspectives. In the first place, decentralized optimization algorithms are usually designed to allow for multiple local training iteration between communication rounds to reduce the total communication cost of the training process (e.g.,[90, 128]); in synchronous star-shaped federated learning the number of participants selected per round is typically limited (e.g., [128]), as well as in peer-to-peer topology the number of neighbours to scatter the updates to is bounded (e.g. bounded to 1 such as in GL [70] or in [179]). Stream compression (e.g., by encoding, quantization and/or sparsification of updates) is typically employed to reduce the per-round communication cost [26, 87, 97, 108, 133, 157, 161, 179, 180]. Furthermore, specific strategies can be crafted accordingly to the peculiarities of the model to train (e.g., by introducing asynchrony between the updating of the neural-network parameters belonging to shallower/deeper layers [35]). Stream compression has been mostly explored in star-shaped federated learning, but similar solutions may be easily adapted in peer-to-peer topology. An orthogonal approach is to improve the communication efficiency by reducing the total communication rounds needed for the model convergence (e.g., implementing distributed variants of SGD optimizers [120, 133, 163]). Or again, communication-efficiency can be architecturally favoured by leveraging MEC [118]. Obviously, combinations of the previous strategies are common.

FedAvg can be seen as a periodic averaging protocol that involves in each round of communication only a random subset of the participants. However, FedAvg (and periodic averaging protocol in general) maintains the same frequency of communication independently from the utility of the specific synchronization, e.g., when all models are approximately equal or they have already converged to an optimum then synchronization may be omitted. Leveraging this observation, authors of [90] propose a dynamic averaging protocol to invest the communication efficiently by avoiding to synchronize models when the impact of such aggregation on the resulting model is negligible. To this end, authors leverage a simple measure, $||w_t^i - r||^2$, for model divergence to quantify the effect of synchronizations; specifically, they measure the divergence of the locally trained model, w_t^i, for the round t at participant i, with respect to a reference model r that is common among all participants, e.g. the last received global model, and compare such divergence with an a-priori chosen threshold to decide whether perform a synchronization.

In [97], two strategies have been proposed to reduce the uplink cost in star-shaped FL (explicitly considering FedAvg as baseline) by means of compression, and they are *structured updates* and *sketched updates*. Such strategies can be combined to further compress the data to be sent from clients to server. The peculiarity of structured updates is that the updates are restricted to have a pre-defined structure, and they are directly trained to fit such structure. Two types of structures are considered by authors: (i) updates are enforced to be a low-rank matrix of rank k, with k being a fixed parameter (low-rank updates); (ii) updates are restricted to be a sparse matrix following a pre-defined random sparsity pattern (i.e., a random mask), thus only the non-zero values along with the seed to generate the pattern have to be communicated. Regarding sketched updates, the full (or structured) update resulting from the local training is approximated, i.e. sketched, in a lossy compressed form. To this end, two (compatible and jointly usable) tools are proposed: subsampling, i.e only a random subset of the (scaled) values of the updates are communicated, and probabilistic quantization. As the reader can note in the continuation, several successive works addressing communication efficiency in decentralized training combine subsampling or sparsification and quantization. Furthermore, supported by empirical evidence, authors highlight the usefulness of applying structured random rotations before quantizing to reduce the quantization error.

Similarly to [97], authors of [26] use a combination of basis transform, subsampling and probabilistic quantization to reduce the server-to-client communication cost of FedAvg.[1]

FedPAQ [157], aside from periodic average of models and partial device participation (standard in FedAvg), leverages quantized updates, i.e. quantizing the weight deltas that are the differences between the received model and the locally-computed models, to enhance the communication efficiency targeting a cross-silo federated setting. The authors

[1]Note that in the work [97] the objective is to reduce the client-to-server communication cost.

propose to use the low-precision quantizer [6]. An accurate convergence analysis for both strongly convex loss function and non-convex loss function is provided. As expected, the fastest convergence rate corresponds with synchronization at each local SGD step; for the strongly convex setting tuning the local SGD iterations in each round as $\tau = o(\sqrt{T})$, with T being the total number of iterations, ensures the convergence of the FedPAQ to the global optimal, while for the non-convex setting the period length τ can grow up to $O(\sqrt{T})$ to reach a stationary point with an overall convergence rate of $O(1/\sqrt{T})$.

Building on their previous Sparse Binary Compression (SBC) [162] technique that targets the traditional distributed setting, in [161] authors specifically design a compression framework for cross-device federated settings. The proposed Sparse Ternary Compression (STC) compresses both the upstream and the downstream communication with respect to the baseline FedAvg while improving the robustness to non-IID data as well as to partial client participation. In addition to experimentally confirming the already known weakness of vanilla FedAvg in presence of heterogeneous data, authors also show poor model accuracy with aggressive quantization schemes, such as SignSGD [15], in non-IID scenarios.[2] Conversely, $top_{p\%}$ sparsification, i.e. dropping all but the p fraction of updates with the highest magnitude, suffers least from heterogeneous data. This observation leads the design of the proposed compression scheme for the upstream communication in FL. As happens in SBC, STC exploits (i) $top_{p\%}$ sparsification of weight deltas (i.e., the difference between the global model and the local model), (ii) local residual accumulation[3], (iii) binary quantization of the $top_{p\%}$ elements[4] and (iv) encoding (to losslessly compress the distance between the non-zero elements of the sparse weight-update) to reduce the amount of data to be sent from participants to the server. It is worth to highlight once more that this strategy alone does not affect the downstream communi-

[2] In SignSGD [15], gradient updates are locally quantized to their binary sign from clients. The parameter server gathers such binary updates and broadcasts the belief about the sign of the true gradient. The server uses majority vote on the gathered gradient updates (See Algorithm 3 in [15]).

[3] Note that, differently from [180] (presented later on), in STC (and SBC) the residual accounts for ignored weights and not for gradients.

[4] The result of the sparse weight-update binarization is a ternary tensor containing values $-\mu$, 0, μ with μ being the mean of the $top_{p\%}$ weight-updates in absolute value. STC sets all the positive non-zeroed elements to μ and all the negative non-zeroed elements to $-\mu$. Note that in SBC the resulting sparse tensor is binary instead, and the algorithm is slightly different; they independently compute the mean of all non-zeroed positive and all non-zeroed negative weight-updates; if the positive mean is bigger than the absolute negative mean, they set all negative values to zero and all positive values to the positive mean and vice versa.

cation. In this regard, authors observe that, although clients-to-server updates are sparse, the server-to-clients update essentially becomes dense as the participation rate, i.e the fraction of participants involved in each round, exceeds the inverse sparsity, i.e. the inverse of the hyperparameter that rules the sparsification. In fact, in the worst case, the number of non-zero elements in the aggregate (the sum) of clients-to-server updates grows linearly with the number of participating clients. The dense nature of server-to-clients updates prevent an effective compression. Therefore, they propose to apply their STC algorithm also to the aggregated updates at server side, hence the server maintains a residual as well. However, the partial client participation in each round of FL prevents a straightforward application of STC at server-side: STC sparsifies and compresses weight deltas, and, considering that not all the participants are involved in every round, some participants could not recover the updated weights from the received (compressed) delta, since they may not have participated to the previous round(s). The solution adopted is to cache the last τ updates at server-side, and to require a prior synchronization step for those outdated participants before initiating the local training. Thanks to this shrewd protocol addition, the downstream communication can be effectively reduced regardless the partial client participation.

In Edge Stochastic Gradient Descent (eSGD) [180], besides tacking advantage of edge servers to scale the collaborative training process, authors propose an algorithm to reduce the uplink communication cost when exchanging gradients in a star-shaped synchronous learning framework. The solution builds on the observation that gradients, produced by iterations of mini-batch SGD optimization, are very sparse [176]; in eSGD, participants upload only a fraction (i.e., a fixed percentage) of the gradient coordinates, only the ones that are considered important, while accumulating a residual to account for ignored coordinates[5] — merely dropping these portions of gradients, even if they are small values, can hamper the model convergence [5].

To reduce the network traffic headed to the cloud, a MEC-aware extension of FL is proposed in [118], namely Hierarchical Federated Averaging (HierFAVG). Authors exploit the hierarchical architecture of such brand-new paradigm to have middle-level ag-

[5]Gradient sparsification and local gradient accumulation is a well-known technique in the traditional distributed setting to reduce the communication cost by speeding up the training process (i.e. less communication rounds) without significantly degrading the resulting model accuracy [5, 117, 176]. Error accumulation, in this case weight accumulation, permits to not waste gradient information, although they may suffer from staleness.

gregator entities; each τ_1 local updates, edge servers gather the updates of the participants in their proximity to produce the aggregated models of their locality; each τ_2 edge-level aggregations, the cloud updates the global model (hence each $\tau_1\tau_2$ local iterations). It is worth noting that if τ_2 is equal to 1, the HierFAVG corresponds to the traditional FedAvg, while, intuitively, with τ_2 greater than 1, HierFAVG reduces the communication cost with respect to FedAvg.

From another perspective, the communication cost of decentralized training can be reduced if less rounds are needed to reach a certain target accuracy. To this end, authors of [133] empirically demonstrate the suitability of an Adam[95]-inspired variant of FedAvg. As well known, the ADAM optimizer leverages per-parameter learning rates, 1st moment and 2nd raw moment estimates to converge faster in traditional minibatch SGD. In the proposed CE-FedAvg, participants locally compute their update by exploiting ADAM, and they send back to the server the 1st and the 2nd moment estimates as well as the locally trained model (specifically, their deltas). Thus, beyond the global model parameters, the server also aggregates the 1st and the 2nd moment estimates, that are broadcasted at the beginning of every round to the learners. Since moment estimates have the same size of model parameters, it is straightforward to note that the communication cost per round is tripled with respect to FedAvg in absence of compression. However, authors highlight that this is compensated by the faster convergence of CE-FedAvg. [6] Furthermore, they employ compression techniques to reduce the amount of data to be sent; sparsification, quantization and encoding are used. Authors also emphasize an additional advantage of CE-FedAvg over FedAvg: in absence of a central test/validation set of data, it is difficult to tune the learning rate for FedAvg, while the default Adam's hyperparameters seem to be suitable for general use.

Similarly, the authors of [120] implement a federated version of momentum gradient descent, namely Momentum FL, where momentum terms and model updates are exchanged between participants and server, round by round, doubling the communication cost of each round with respect to FedAvg, while taking advantage of faster convergence rate.

[6]Note that CE-FedAvg implements a distributed version of the Adam optimizer, which is different from the approach in [155], presented in detail in Chapter 7, that instead uses Adam as a server-side optimizer that sees the aggregated model updates as "pseudo-gradients". Furthermore, in FedAdam [155] the per-round communication cost is the same as FedAvg.

The same purpose, i.e. reducing the total communication rounds to reach model convergence, motivates an improvement of the CFA algorithm (already presented in 14) in peer-to-peer topology of learners. Authors propose to introduce a "negotiation" phase where, before using the aggregated model ψ_t^k to run local training, the participant k feeds back ψ_t^k to the same neighbors. Neighbors compute gradients with respect to ψ_t^k, and send them back to the participant that has forwarded the request. Next, gradients are aggregated, leveraging a tunable mixing parameter, to produce $\widetilde{\psi_t^k}$ that is then used as starting point for the local learning iteration. This strategy should make the learning faster[7]. However, this algorithm requires four communication rounds, and moreover the negotiation is synchronous. Therefore, the algorithm is transformed into a two-stage algorithm, referred as Consensus FedAvg Gradient Exchange (CFA-GE) [163]: the negotiation phase is performed without the need of sending ψ_t^k and receiving back the neighbors' gradients, permitting to save communications and avoid the synchronization intermediate step (i.e., waiting for the neighbors to send back the gradients with respect to ψ_t^k). The insight is to exploit past (and outdated) models received from a certain neighbor during the previous rounds to produce, in advance, a gradient prediction for that neighbor, and this is done for all the neighbors. In this way, it is possible to scatter such gradients prediction together with the next-generation model parameters; each participant hence receives such information, produces ψ_t^k by aggregating the neighbors' model as we have seen for the baseline CFA algorithm, and uses the received gradient predictions to adjust the model to obtain $\widetilde{\psi_t^k}$, and finally applies the local training to $\widetilde{\psi_t^k}$ that will generate the updated model.

In [179], the authors propose an efficient peer-to-peer framework for cross-silo communication, namely SAPS-PSGD, where aggressive model sparsification is coupled with single-peer communication scheme. They leverage a coordinator entity – not a parameter server – that, in extreme synthesis, broadcasts to the participants a gossip matrix and other some necessary information (i.e., the current global step, a random seed to generate the mask for applying the desired sparsification) and synchronizes the rounds of communication among such node pairs. The gossip matrix is built by taking into account the peers' bandwidth to favour faster links; it dynamically determines the couples of peers that will exchange highly sparse model updates during that round.

[7]The negotiation phase, from an high-level perspective, can be thought to be similar to the approach of [109].

3.2 Protecting Pivacy

It may be believed that sharing gradients, model updates or meta-level information (such as outputs of layers in neural-networks) in place of raw data ensures privacy protection. However, it has been demonstrated that gradients exchanged during the distributed training process do leak information about the training data [63, 72, 140, 148, 216, 227] as well as model updates [131, 140] — even though it may be preferable to exchange model weights instead of gradients under a privacy-preserving perspective [149] — and activations [44, 205].

The literature about protecting privacy in decentralized learning comprises diverse approaches; differentially-private mechanisms [54, 129] can be employed during the distributed training process to mask updates at the cost of reduced model accuracy [11], and relaxations of traditional Differential Privacy (DP) can be leveraged to inject less noise [183], limiting the incurred performance degradation. Data-augmentation [52] and obfuscation [74] techniques can be used in visual application to prevent reconstruction of images in the training set. Multi-party secure aggregation [19, 172] and similar techniques [63] can hide the individual contributions to the aggregator, finding its main utility in star-shaped federated learning, but producing non-negligible overheads. Additively homomorphic encryption also allows the aggregator to sum updates, thus ensuring the inscrutability of single contributions [148] while not degrading model accuracy but increasing communication cost. Combinations of DP-mechanisms with secure aggregation and additively homomorphic encryption are also explored [60, 185] to balance the weaknesses of such techniques. Minimizing distance correlation between raw data and activations (at cut layer) [187] and step-wise activation functions [205] are used to prevent the invertibility from intermediary representations in the context of privacy-preserving Split Learning.

The first works enforcing participant-level (ϵ, δ)-DP [48] in federated settings are most notably [54] and [129]. The aim, common to both the works, is to ensure that a model trained with FedAvg does not reveal whether a certain participant has been involved during the decentralized training process, balancing the trade-off between privacy loss and model performance. It is worth highlighting that the proposed solutions protect the whole client's dataset differently from [1] where a single data point's contribution in the trained model is protected.

Authors of [54] use two randomized mechanisms to guarantee client-level DP: (i) random subsampling of participants for a certain round of communication; (ii) Gaussian mechanism. In FedAvg, the central aggregator averages the participants' updates, that here are considered to be weight deltas (i.e., the difference between the received parameter weights and the locally computed parameter weights). The key idea of [54] is to perturb and approximate such averaging (i.e. perturbing the sum of updates) by employing a Gaussian mechanism. As usual, the Gaussian-distributed noise has to be calibrated according to a certain sensitivity; such sensitivity is calculated as the median norm of all the gathered updates and the updates are scaled according to such sensitivity, i.e. clipped updates.[8] To keep track of the privacy loss within subsequent communication rounds, authors use the moments account of [1] instead of the privacy amplification lemma and the standard composition theorem [48] to obtain tighter bounds. In particular, they stop the collaborative training once the (cumulative) δ, that represents the likelihood that a participant's contribution is disclosed, becomes greater than a threshold.

The approach of [129] is slightly different from [54]. Authors, in fact, randomly sample participants by selecting each independently with probability q, hence producing variable-sized samples of participants and influencing the sensitivity of (weighted) average queries — in [54] a fixed number of clients is randomly selected. Two different bounded-sensitivity estimators are proposed to account for such participant-sampling process. Furthermore, two clipping strategies are evaluated for multi-layers models: (i) flat clipping, i.e. using an overall clipping parameter, or (ii) per-layer clipping, i.e. treating the parameters of each layer as separate vector and using per-layer clipping parameters, motivated by the observation that such vectors may have vastly different L_2 norms — anyway the clipping parameter is fixed throughout the training process, while in [54] is dynamically calculated as the median norm of all the unclipped contributions.

In [183], authors allocate a tighter privacy budget for guaranteeing client-level DP and instance-level DP, i.e. less noise to reach the same privacy guarantee, also improving the accuracy of the trained model. They employ a relaxation of traditional DP, in this case Bayesian DP (BDP) [184], by making two assumptions (i) stationary data distribution and (ii) datasets with unchangeable samples. Authors also use a Bayesian accounting method instead of state-of-the-art moments accountant [1] thanks to the assumption that data

[8]The sensitivity is calculated by the server in each communication round.

come from a particular distribution and not all the data are equally likely; this observation can lead to sharper privacy loss bounds with BDP in federated setting. Besides the proposed use of BDP, to limit the noise added to guarantee both instance-level and client-level DP, the noise to be added by the server for client-level DP is "re-counted" considering the injected noise during the on-device gradient descent. They call this approach joint accounting. However, a limitation emerges: joint accounting is only usable for FedSGD algorithm, not for FedAvg (because the possible multiple local iterations in FedAvg, hence multiple noisy steps, may influence the point at which the gradient is computed: a different gradient distribution can arise or the total noise variance can be underestimated).

To prevent the server from peeking in individual updates during the aggregation phase, a practical protocol for secure aggregation, namely SECAGG, has been proposed in [19] for federated settings — reminding that the communication bottleneck and the dropping of users are peculiar of such scenarios. In a nutshell, star-shaped FL systems leverage a central server that computes sums of updates from which deriving the new-generation global model round by round. The scope of SECAGG is to hide the individual contributions of participants and release only the sum of such updates to the server, preventing privacy violations from the aggregator entity. The essence of the approach is similar to differential privacy: updates are locally perturbed, but, while in DP-mechanisms such perturbations become part of the updates (they are never removed, in fact noise calibration is fundamental to not compromise the training), in SECAGG such perturbations are neutralized during the aggregation phase. The insight is to have pairs of participants — hereinafter referred as participant u and participant v — that share randomly sampled 0-sum pairs of mask vectors, $p_{u,v}$ and $p_{v,u}$; before uploading their model updates, participants u and v add such masks to their contributions, with $p_{u,v} + p_{v,u} = 0 \, \forall u \neq v$; each participant u computes a random mask vector and perturbs (i.e., adding $p_{u,v}$ if $u > v$ or subtracting $p_{u,v}$ otherwise) its local update for each other user v; mask-pairs are canceled out during the sum of all contributions. Every pair of participants share a common random seed $s_{u,v}$ of some fixed length that can be fed to a secure Pseudorandom Generator PRG [17] to generate the mask pairs, hence the seed can be transmitted in place of the the entire mask (that has the same size of updates) reducing the communication burden. These shared seeds are established through Diffie-Hellman [42] key exchange, composed with a hash function. It is worth noting, that (i) SECAGG requires the elements of the input vectors, i.e. the participant's updates, to be integers $modK$, while (ii) the elements of the vector updates are typically real-valued instead, and that (iii) the employed PRG's output space is the same of the input space. Therefore, the real-valued elements of the updates are typically clipped to a fixed range of real numbers, and then quantized among such range using k bins, and the SECAGG modulus is chosen to be $K = kn$, with n being the number of participants.

A practical protocol for collaborative training in federated settings must be able to tolerate a fraction of dropping users. To this end, SECAGG leverages Shamir's t-of-n Secret Sharing [166] to permit recovering the pair-wise seeds of a limited numbers of dropping participants; in practice, each participant sends encrypted shares of its Diffie-Hellman secret to all other participants via server. SECAGG also accounts for the critical case in which a certain participant belatedly responds to the server with its contribution by using a double masking for the updates. In addition to $p_{u,v}$, a private mask vector p_u (generated from a seed b_u as well) is further added to the update, and also its shares are distributed during the secret sharing round for the pair-wise masks.

SECAGG has been employed in the FL system designed in [18] but highlighting that the quadratically grow (with respect to the number of participants) of the computational cost for the server limits the maximum size of an instance of SECAGG to hundreds of learners. They indeed leverage intermediate secure aggregators for subsets of participants, and the intermediate sums are further aggregated without SECAGG by a master aggregator.

A recent work [172], namely Turbo-Aggregate, addresses the quadratic growth of the computational cost and of the communication overhead by slightly changing the approach, and still being resilient to user dropouts (up to 50% of participants). The key idea is to partition the federation of learners in groups that actively participate in the aggregation and dropout-recovery phases instead of just leveraging the central server, and to add redundancy directly in the model updates to reconstruct the missing contributions of dropout participants instead of Shamir's t-of-n Secret Sharing such as in SECAGG. In a nutshell, reminding that the scope is to securely compute a sum (i.e., the sum of locally computed updates) and assuming that all communications take place via central server employing Diffie-Hellman key exchange protocol, Turbo-Agg works as follow. Firstly, participants are randomly divided in L groups, with each group being composed of N_l participants. The set of participants in group l is referred as U_l. The process involves L stages, and Turbo-Agg adopts a circular and sequential strategy in its simplest version: in each stage only one group is involved; the output produced from a group in a certain stage is the input for the next group.[9] Ignoring for a moment the possibility of dropout, in each stage, the participant i in group l masks its update $x_i^{(l)}$ with a random vector $u_i^{(l)}$ being known (and communicated) only by the honest server, similarly to what happens in SECAGG. To be secure against server-participants collusion, learner i additionally masks its update with another random vector $r_{i,j}^{(l)}$, and the resulting masked update $\tilde{x}_{i,j}^{(l)} = x_i^{(l)} + u_i^{(l)} + r_{i,j}^{(l)}$ is sent to each participant j of the group $l+1$, with $\sum_{j\in[N_{l+1}]} r_{i,j}^{(l)} = 0$, i.e. random vectors r cancel out during aggregation. The secure sum is cooperatively computed, group by group, and can be summarized thanks to the recursive relation $\tilde{s}_i^{(l)} = \frac{1}{N_{l-1}} \sum_{j\in[N_{l-1}]} \tilde{s}_j^{(l-1)} + \sum_{j\in[U_{l-1}]} \tilde{x}_{j,i}^{(l-1)}$ with $\tilde{s}_i^{(l)}$ that is a variable locally held by each participant i in group $l > 1$, and that represents the aggregated masked updates from the previous group[10]. It is important to highlight that each participant i of group l sends $\tilde{s}_i^{(l)}$ and $\tilde{x}_{i,j}^{(l)}$ to each learner j of the group $l+1$. A final aggregation step is necessary to preserve the privacy of the participants in group L at the stage L; an additional group (referred as $final$), in fact, is randomly composed (for example, among the survived learners) with each participant aggregating the contributions coming from the group L, and sending the results to the server. Specifically, participants j in the $final$ group produces $\tilde{s}_j^{(final)} = \frac{1}{N_L} \sum_{i\in[N_L]} \tilde{s}_i^{(L)} + \sum_{i\in[U_L]} \tilde{x}_{i,j}^{(L)}$ and send it to the server, that can recover the sum of unperturbed updates by applying $\frac{1}{N_{final}} \sum_{j\in[N_{final}]} \tilde{s}_j^{(final)} - \sum_{m\in[L]} \sum_{j\in[U_m]} u_j^{(m)}$. However, in case of participant dropouts the protocol will fail, since, for example, the random vectors r cannot be cancelled out. To this end, authors propose to employ Lagrange coding [206] to allow participants of group l to recover the missing contributions from group $l-1$, and to compute the partial aggregation anyway. Being concrete and redirecting to the full paper [172] and to [206] for theoretical detail, each participant has to send to each participant j in group $l+1$ two additional (coded) vectors in each stage, namely $\bar{s}_i^{(l)}$ and $\bar{x}_{i,j}^{(l)}$, in addition to $\tilde{s}_i^{(l)}$ and $\tilde{x}_{i,j}^{(l)}$. The employed coding strategy allow each learner in group $l+1$ to reconstruct the vector $\{\tilde{s}_i^{(l)}\}_{i\in N_l}$ starting from at least N_l evaluations (i.e, $\bar{s}_i^{(l)}$ and $\bar{x}_{i,j}^{(l)}$) from the previous stage. Therefore, since each participant send two evaluations to the learners in the next group, this redundancy permits to tolerate up to half of learners dropping.

[9]Since only one group is active per stage, for ease of notation, group and stage are referred both with the index l.

[10]The initial aggregation at group $l = 1$ is set as $\tilde{s}_i^{(1)} = 1$.

It is worth noting that, although SECAGG and its variant Turbo-Aggregate explicitly targets star-shaped networks of learners, they are suitable for fully decentralized networks, i.e. peer-to-peer topologies, with one peer (or more) working as aggregator.

An alternative to SECAGG for star-shaped FL frameworks is represented by Additively Homomorphic Encryption; since such technique guarantees the additivity of multiple ciphertexts, the server can perform the aggregation without the need of seeing the updates in clear. In [60], authors propose to use a symmetric additively homomorphic encryption called PPDM [223] for its efficiency, combining it with Laplacian mechanism for DP in order to neutralize collusion between compromised users and malicious server. They show drastically reduced communication overhead with similar solution [148], that employs paillier encryption instead.

In [185], authors combine multi-party computation (MPC) via Threshold Homomorphic Encryption and Differential Privacy to balance their respective weaknesses; in fact, applying DP to provide the required level of privacy may degrade accuracy while MPC alone is vulnerable to inference attacks over the output, i.e. the intermediate models during the collaborative training process and the final predictive model. Leveraging only on one of those two techniques may compromise the effectiveness of the system (in terms of prediction accuracy of the resulting model or in terms of privacy guarantee). The key intuition in [185] is to reduce the traditional amount of locally-injected noise to ensure ϵ-DP by exploiting the MPC framework building on the assumption that t participants are trusted (i.e., non-colluding parties), with t being a customizable parameter; thanks to this assumption, the Gaussian noise to be added to each local query is reduced by a factor of $t-1$. In the worst scenario, the performance (in terms of model accuracy) of the proposed system converges with existing local DP approaches.

Considering the scenario in which the data quality of certain participants, namely *unreliable participants*, may be poor (meaning that a portion of their data is not always accurate as the data held by others), authors of [218] focus on guaranteeing two levels of privacy: (i) preserving privacy of the participant's data and (ii) hiding the eventual participation in the training process of unreliable participants. At the same time, they focus on limiting the impact on the global model of such participants. The proposed solution, SecProbe [218], ensures participants' privacy by perturbing, during the local training process, the objective function of the neural network using the functional mechanism

(FM) [212] to achieve ϵ-DP, and obtaining the sanitized parameters by minimizing the perturbed objective function.

To make the metadata exchanged in Split Learning irreversible, in [205] authors propose to modify the conventional activation functions to be step-wise, i.e. the activation function is discretized by having the input domain divided into intervals and the output constant for each interval; in this way, it is not possible to exactly recover the activations' input from their outputs[11]. In this context, another approach to reduce invertibility of intermediate representations consists in minimizing the distance correlation between raw data and the communication payload, i.e. having a low distance correlation while maintaining the accuracy in predicting the output labels. Authors of [187] hence train the neural network by using a weighted combination of two losses as loss function, and such losses are the log distance correlation [177] and the categorical cross entropy. The former is used as a measure of statistical dependence between the input data and the estimated cut layer activations, while the latter traditionally considers the true labels for the inputs and the predicted labels. Intuitively, the distance correlation is minimized to ensure privacy and the cross entropy is minimized for classification accuracy. The solution is evaluated on visual datasets.

3.2.1 Combining Privacy and Communication Efficiency

Lossy compression techniques inherently lead to a privacy improvement, however it is not straightforward to measure the effective privacy guarantees, for example under DP formalism. The works surveyed in Section 3.1 do not explicitly measure privacy, and the ones in Section 3.2 do not address the communication cost as primary concern, while examples of combined approaches can be found in [108] and in [87]. Furthermore, other aspects in conjugating privacy and communication efficiency emerge; the secure aggregation protocol [19] can be redesigned to account from the beginning for communication efficiency [20], while tailored DP-mechanisms can be more amenable to privacy analysis when quantization of noisy DP-updates is employed [4].

[11]Authors of [205] consider three activation functions: sigmoid, hyperbolic tangent and ReLU [138]. While sigmoid and hyperbolic tangent are bijective functions, ReLU is a surjective function, and the output of ReLU can be reversed only if the input is positive. The proposed solution "masks" the output of such positive inputs by using a step-wise variant of ReLU.

In [87], authors combine communication efficiency, privacy guarantees and resilience to malicious participants under non-IID data distribution. They consider a star-shaped synchronous collaborative learning framework in which participants and server exchange (aggressively compressed) gradients instead of model parameters. The proposed algorithms use as baseline the SignSGD [15] algorithm with majority vote, that, however, does not explicitly and formally address privacy protection of participants and that has been shown to fail to converge when the data on different learners are heterogeneous [34, 161]. In particular, to deal with non-IID data, authors first propose a variation of SignSGD, namely *sto-sign*, that applies a two-level stochastic quantization on locally computed gradients, and then only transmits the signs of such quantized values. Additionally, *dp-sign*, a differentially private version of *sto-sign*, is designed to ensure formal privacy guarantees for participants involved in the training. Authors theoretically relate the Byzantine resilience, i.e. the number of Byzantine workers that can be tolerated without harming the convergence guarantees, of their proposed algorithms to the heterogeneity of local datasets. [12] Authors also propose an extension of their algorithms which takes account for residual error on server side and uses it to correct the majority vote. The convergence of the proposed algorithms is established theoretically.

With respect to just sending the quantized updates in clear, the SECAGG[19] protocol leads to a bandwidth expansion that is less than 2x while ensuring reliability of the secure aggregation to dropping or collusion of a fraction of users.[13] However, in [20], authors critically observe some limitations of a straightforward combination of SECAGG and compression techniques; chief among them (i) quantizing to a fixed point representation requires selecting the clipping range $[-c, c]$ a priori that may be challenging to establish or may lead to poor approximations if the clipping range is not large enough, and (ii) the SECAGG modulus is chosen to be $K = nk$ to represent all possible aggregated vectors without overflow (for example, if clients are 2^{10} the SECAGG modulus are 10 bits wider than they would be without accounting for secure aggregation) dominating the communication cost introduced by SECAGG — the bandwidth expansion determined by secret

[12]A Byzantine participant may transmit arbitrary information. Authors of [87] assume that such Byzantine participants upload the opposite signs (the opposite sign of each entry) of the true gradients, with the true gradients being the average gradients of all the normal workers (hence, it is supposed that the attackers know such quantities).

[13]1.73x bandwidth expansion considering 2^{10} participants (i.e., $n = 2^{10}$) and 16 bit fixed point representation (i.e., $k = 2^{16}$).

sharing and cryptography is much less influential. The scope of [20] is to propose a recipe for an auto-tuning (observation (i)) communication-efficient (observation (ii)) secure aggregation. The key idea is to avoid clipping at client-side but instead quantizing over an unbounded range according to a quantization bin size b that is dynamically and tightly adjusted by the server (and communicated round by round) according to the distribution of the entries of the sum relative to the previous round, and then locally applying the $mod\ k$ operation instead of clipping; the server can compute a tight bin size b exploiting the assumption that the entries of the sum fit a normal distribution thanks to a random rotation that is locally performed by the participants (before quantizing) to their updates.

3.3 Data Heterogeneity and Client Drift

While in a typical distributed DL setting (e.g., a datacenter environment) training data on worker machines are assumed to be independently and identically distributed (IID), such assumption does not hold for FL settings: the local data on clients are likely not to be representative of the global distribution (i.e., the union of local datasets).

Considering a classification task, data heterogeneity among clients can be determined by (1) label distribution skew, (2) feature distribution skew and (3) quantity skew. In label distribution skew, the label distributions vary among participants. This can be very common in reality, thinking for example to the health care domain, where a specific hospital may treat patients with specific diseases/pathologies, collecting data examples for a subset of the total possible class of diseases/pathologies.

In literature, the label distribution skew is reproduced leveraging two major strategies, (a) quantity-based label imbalance, where clients hold examples belonging to a fixed number of labels – in practice, the original dataset is partitioned in subsets of a certain size, with each subset containing only examples with the same label, and each participant is assigned a certain amount of per-label subset, (b) distribution-based label imbalance, where the per-client label distributions is ruled by a distribution. In the latter case, usually the label imbalance is drawn from a Dirichlet distribution, as proposed in [76]. In this case, the degree of heterogeneity, i.e. how much the amount of per-label examples vary across clients, is tuned by a (non negative) concentration hyperparameter – the lower the concentration the higher the heterogeneity, with two extreme cases: with large concentration values (e.g. > 100) clients will hold approximately the same amount of per-label examples, while with low concentration values (e.g., < 0.1) client will tend to hold data belonging to only one class.

In feature distribution skew, the focus is on the distribution of features while the amount of per-class examples may be the same. For example, in a real federated visual dataset, images belonging to the same class may exhibit different features for different clients (e.g., the same bird may vary in feather color and pattern depending on the specific habitat). The work in [105] identifies two ways of generating datasets with a feature distribution skew, i.e. noise-based feature imbalance and synthetic feature imbalance. In the first case, different levels of noise (e.g., Gaussian noise) is injected to the examples of different clients, which can be created by randomly and uniformly partitioning a balanced IID dataset. An example is depicted in Fig. 3.2. For the latter case, synthetic feature imbalance refers to dataset explicitly generated to have different features in the same class of examples, to be then distributed on different clients.

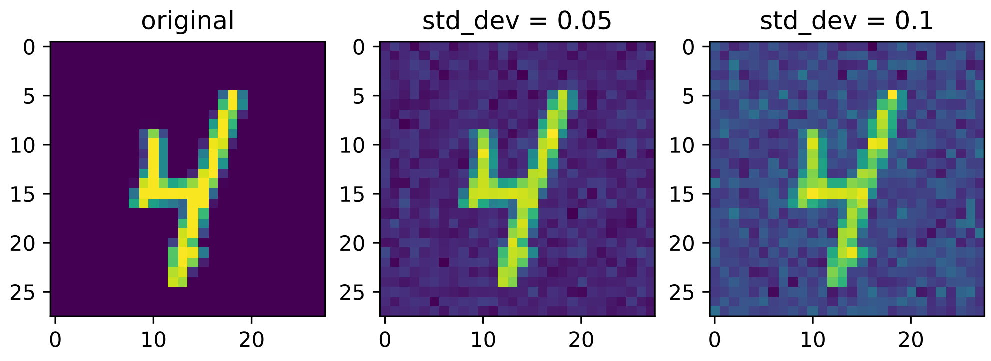


Figure 3.2: Example of adding Gaussian noise, with varying standard deviation, on an image from the MNIST dataset.

In quantity skew, the amount of local data examples varies across participants; also here, it can be regulated by a Dirichlet distribution, similarly to the distribution-based label distribution.

It is worth noting that these three kinds of heterogeneity, i.e. label distribution skew, feature distribution skew and quantity skew, can be present simultaneously in real-life applications. For example, in the FEMNIST dataset [25], a federated version of the MNIST dataset, each client holds the handwritten digits and letters from the same person, hence

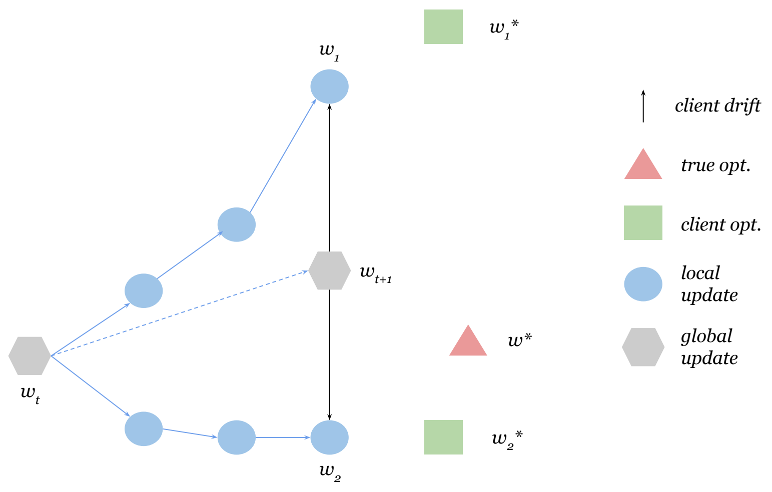


Figure 3.3: Client drift phenomenon in FedAvg is depicted for two learners performing three local updates (light blue circles). Such local steps move towards the individual client optima w_i^* (green square) by minimizing their cost function for local data. The server aggregates client updates by average, and therefore the updated version of the global model at the next round (i. e., w_{t+1}) (gray hexagon) move towards the average of client optima instead of to the true optimum w^* (red triangle) [91].

retaining his or her unique style (feature distribution skew), and each client holds a different amount of per-label examples (label distribution skew) and of total examples (quantity skew).

Non-IIDness of local data implies that the local objective of each FL participant is inconsistent with the global optima. In fact, when fine tuning on private data, client model drifts apart from the global model they received at the beginning of the round and also from the models of the other clients, which hold data drawn from a different distribution. Such a phenomenon is referred to as client drift, depicted in Fig. 3.3.

It is worth underlining that client drift is more significant in presence of large number of local updates (large number of epochs and/or small local batch size) [91, 110, 189], which amplifies the divergence among personalized client models, while, on the other hand, a too reduced number of local iterations translates to high cumulative communication cost (i.e., a high number of rounds is needed to achieve global model convergence).

For the sake of clarity, Fig. 3.4 reports some qualitative results to highlight the degradation, in terms of global model generalization, introduced by non-IIDness. The accuracy of a global model trained while considering three different levels of data heterogeneity (low, moderate, and high) is tracked, empirically showing that a higher level of data heterogeneity translates into reduced performance of the FedAvg global model (test accuracy). The data among clients is allocated following a distribution-based label imbalance (case 1b above), and ruled by a concentration hyperapameter α (larger the α the higher the heterogeneity level). Fig. 3.5 illustrates the distribution of labels among clients.

In the last Chapters of this Thesis, we will focus on state-of-the-art methods to tackle data heterogeneity with algorithmic approaches. In Section 6.3.2, we will review FL approaches, based on Knowledge Distillation, that tackle data heterogeneity either rectifying the server model resulting from averaging drifted client models or directly limiting client drifts during local training. In Section 7, we will review and discuss the state-of-the-art algorithms to tackle data heterogeneity from a general perspective, and we empirically evaluate a selection of the reviewed methods.

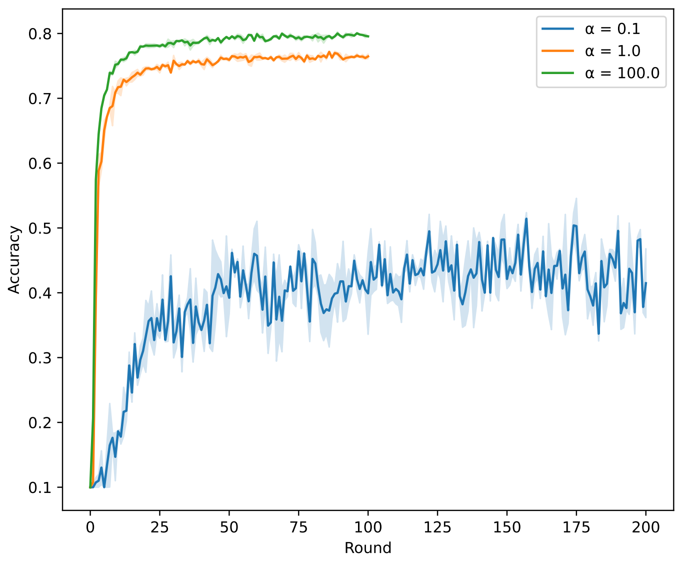


Figure 3.4: The figure shows how the accuracy of the global model trained with FedAvg deteriorates with increasing data heterogeneity. A lower α coefficient translates to higher non-IIDness. The performed experiments consider the CIFAR-10 dataset, and a ResNet-18 architecture.

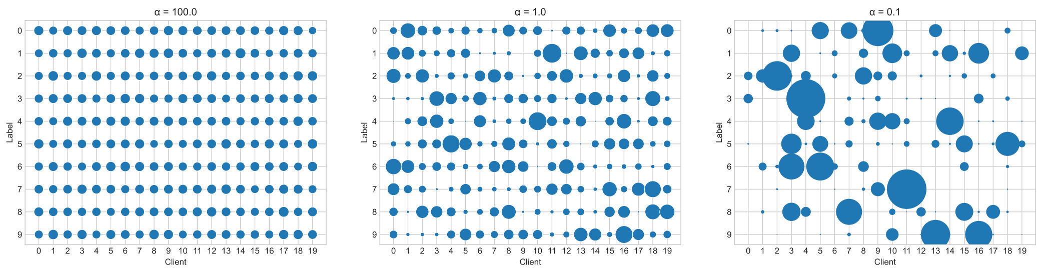


Figure 3.5: The figure shows the distribution of labels (0-9) on the private data of 20 clients. From left to right the heterogeneity is increased by reducing the α parameter that rules the Dirichlet distribution, as in [76]. This is an example of label distribution skew, simulated with distribution-based label imbalance.

3.4 Handling System Heterogeneity

Device heterogeneity, i.e. device with diverse hardware characteristics or/and with different connectivity (in general referred as *resources*), is common in cross-device federated settings. Such heterogeneity negatively influences the training process; for example, in federated learning frameworks that leverage synchronous rounds, the slower participants dictate the pace if any counteraction is taken.

Inspired by the well-known dropout technique [174], clients train their updates considering a smaller sub-model with respect to the global model. This further reduces the server-to-client traffic, reduces the local computational cost, shrink the local memory requirements, and obviously, reduces the client-to-server traffic. Differently from the traditional dropout, a fixed number of activations are zeroed out at each fully-connected layer, thus all the possible sub-models have the same reduced architecture, while a fixed percentage of filters are zeroed out for convolutional layers. Authors call this strategy Federated Dropout. As we will detail in Chapter 5, Federated Dropout can be extended to allow for differently-sized sub-models and can be coupled with lossy and lossless compression.

From a different perspective, authors of [196] claim that the synchronous nature of FedAvg can limit the scalability, the efficiency and the flexibility of the FL framework. In fact, (i) only few hundreds of participants are selected per round due to avoid server-side congestion (the server broadcasts the global model at the beginning of every rounds to all the selected participants); (ii) given the heterogeneity of training devices (e.g., there could be significant diversity in terms of computational power), the server usually sets a timeout for receiving back the updates and then synchronizing the model. It could happen that the selected participants that are able to complete the round within such timeout are not enough to produce a reliable update (i.e., less than the minimum participant goal count) [18]. By leveraging asynchronous updates, FedAsync avoids server-side timeouts and abandoned rounds as well as not requiring to broadcast the model to all the selected participants at the same time. Moreover, to limit the effect of staleness, a well-know drawback of asynchronous SGD approaches, FedAsync uses a weighted average to generate the new global model after aggregation as happens in SLSGD, relying a mixing hyperparameter that weighs the freshness of the aggregated model. Furthermore, to deal with drifting clients and non-IIDness, a proximal term in the local objective functions is employed as it happens in FedProx. Different alternatives are proposed to account for staleness, and to adaptatively decrease the mixing hyperparameter that rules the average in function of staleness, i.e. less weight associated with larger staleness.Under the same communication overhead, they show that FedAsync converges fester than FedAvg when staleness is small while the two approaches have similar performances considering large staleness for FedAsync. Authors state that, in general, the convergence rate of FedAsync is between single-thread SGD and FedAvg.

Asynchronous approaches, such as FedAsync [196], limit the influence of resource-constrained devices on the collaborative training process — synchronization among participants requires to wait for the slowest. In TiFL [28], authors design a system to alleviate the stragglers problem without relaxing the synchronization of FedAvg, but by clustering participants in tiers with similar response latency per round, while in LoAdaBoost [79], authors propose to use the cross-entropy loss information to early stopping the local training.

Besides asynchronism and tier of participants with similar response latency, a natural solution to address straggler clients in FL frameworks (resource constrained devices and/or devices under poor network condition) was priorly proposed in [142], in their FedCS. The goal is to maximize the number of updates to be aggregated within a specific deadline, since involving a larger fraction of participants in each round typically reduces the time needed to achieve a certain model accuracy [128]. Taking advantage of the MEC infrastructure, authors propose to extend the FL algorithm by replacing the random selection of clients with a two-step client selection; the MEC operator asks random clients to provide their resource information (computational capacities, wireless channel states, size of the dataset relevant to the current training task) from which deciding whether including them in the current training round according to an estimation of the time necessary for such participants to complete the download-train-upload process.

In [190], authors address the problem of dynamically adapting the global aggregation frequency (in real time) to optimize the learning process with a given resource[14] budget targeting a star-shaped FL framework in edge computing environments. They consider M types of resources that can be taken into account, and define that all the participants consume c_m units of type-m resource at each local update step, and each global aggregation consumes b_m units of type-m resource (with $c_m > 0$, $b_m > 0$). Being T, the number of total local update steps for the training process, and being τ, the number of local updates between two global synchronizations, and considering the resulting number of global synchronizations K, i.e. $K = T/\tau$, the total amount of consumed type-m resource is $(T + 1)c_m + (K + 1)b_m$, noting that the additional "+1" accounts for computing the last loss value after the last synchronization K. The objective is to minimize the global loss function by tuning τ and K (and, consequently, T) such that the total amount of consumed type-m resource is not greater than the resource budget R_m (each type-m resource has a certain budget associated). Such minimization problem is approximately solved by leveraging a theoretical convergence upper bound of the canonical distributed gradient descent after T iterations, although assuming that the loss function is (i) convex, (ii) ρ-Lipschitz and (iii) β-smooth. In the convergence analysis, authors also define an upper bound for gradient divergence, i.e. an upper bound of the divergence between the gradient of the local loss function and the gradient of the global loss function, that depends on how the data is distributed among different participants, hence taking into account the non-IIDness of data. We redirect to the full paper for the complete theoretical analysis. In a nutshell, the proposed control algorithm recomputes the optimal τ, hereinafter referred as τ^*, during each aggregation step via linear search on integer values of τ accordingly to the most updated parameter estimations needed to approximately solves the minimization problem mentioned above. [15]

[14]Authors of [190] consider a general definition of "resources" including, e.g., bandwidth, energy, time and monetary cost.

[15]It is worth noting that, intuitively, if the resource budget is unlimited, τ^* is equal to 1, i.e. global synchronization after each local update, while in presence of budget constraints it may be convenient investing the resource for local computations rarefying the global synchronizations, i.e. $\tau^* > 1$.

In regards to peer-to-peer frameworks, BACombo (already presented in 14) interestingly leverages a bandwidth-aware worker selection, i.e the peers to be requested for model segments are not trivially chosen randomly. To reduce transmission time, peers with faster network connections should be preferred. However, it is not easy to know the network condition of a certain peer a priori. The proposed solution exploit a multi-armed bandit algorithm [9]; each participant, with probability ϵ, either explores the network conditions of peers by selecting them randomly or exploits its already acquired knowledge — each participant maintains a table, that is updated each time a peer is picked for communication, that contains historical indications about the network state of that peer — by greedily selecting the peers with best network conditions.

3.4.1 Defending against Poisoning

From being passive data providers, in cloud-based ML, participants become active entities in the learning process of decentralized training: they locally compute updates and observe intermediate model states. Although this design is the cornerstone to improve several aspects of traditional ML/DL, it exposes the system to a larger variety of attacks from malicious learners, since participants, in theory, can contribute with arbitrary updates, and could try to manipulate the learning process for diverse scopes (e.g., merely hampering the convergence, forcing other participants to over-expose their contribution or backdooring the system), while making their detection harder since the raw data are not accessible. This is known as model poisoning, besides the more traditional data poisoning. We redirect the reader to [125] for a complete understanding of the threat model and of the attack variety. We present here some strategies to detect and/or neutralize poisoning attacks.

Authors of [197] (SLSGD) propose a variation of FedAvg to address non-IIDness and to tolerate data poisoning attacks (evaluated by simulating the attack through label flipping). They act on the baseline FedAvg algorithm by varying (i) the aggregation step and (ii) the new-model generation step; (i) instead of aggregating the updates by averaging, they use a trimmed mean to (try to) filter out poisoned updates, and (ii) instead of replacing the previous global model with the resulting aggregated model, they use a moving average between the previous and the just aggregated model to limit the influence of non-IID datasets and to mitigate the extra variance caused by such “robust” aggregation.

In [53], authors propose a defense against sybil-based poisoning (precisely, label-flipping and backdoor poisoning), namely FoolsGold, targeting a federated learning framework where participants upload locally computed gradients to the (honest) aggregator. The idea is to identify malicious colluding participants, i.e. poisoning sybils, by monitoring the diversity of participants’ update; sybils are supposed to share a common objective and the directions of poisoning gradients should seem unusually similar respect to updates from honest learners. In a nutshell, FoolsGold maintains an historical aggregate of updates per participant at server side, i.e. the cumulative sums of its updates so far, and it measures the cosine similarity between couple of participants’ historical aggregates before each aggregation step — the rational behind this strategy is that gradients resulting from single local iteration of SGD can be very similar in directions even among honest clients, however colluding parties will share the same objective in the long run, limiting the effectiveness of poisoning throughout the training process by accordingly re-scaling the learning rate of participants that are deemed as possible sybils. The clear limit of FoolsGold — apart from being incompatible with secure aggregation and assuming honest aggregator — is that it is designed to look for sybils, hence a single participant adversary can remain undetected.

Authors of [217] propose a defense against poisoning, specifically targeting label flipping and semantic backdoor attacks, in a synchronous federated learning framework accounting also for non-IIDness. Differently from FoolsGold [53], their strategy actively leverages on clients; the server asks to the participants to evaluate some sub-models, each one derived from the aggregation of disjoint subsets of the model updates related to a certain round, and they provide back to the server an indication about the correctness in the classification task of such sub-models, tested on their private dataset, in the form of a binary matrix (obviously, a certain participant cannot receive a sub-model derived from its own contribution). Thanks to the gathered matrices, the server computes a penalizing coefficient for each sub-update to weigh the aggregation of such sub-models (for example, if more than half of the clients report the anomaly for the same sub-model, it should be zero-weighed). Authors highlight that their solution can be also combined to FoolsGold [53], e.g. to detect single-participant attack.

Similarly to [53] and [217], authors of [107] use a server-side pre-trained autoencoder model to detect abnormal weight updates that are then accordingly penalized during the aggregation.

4 Structured Sparse Ternary Compression for Convolutional Layers

In this Chapter, we introduce our original variant of STC that has been specifically designed and implemented for convolutional layers, and previously presented in [136]. As for the method in Chapter 5, we devise a star-shaped synchronous protocol which exchanges model parameters and model updates as FedAvg. Our variant is originally based on the experimental evidence that a pattern exists in the distribution of client updates, namely, the difference between the received global model and the locally trained model. In particular, we have experimentally found that the largest (in absolute value) updates for convolutional layers tend to form clusters in a kernel-wise fashion. Therefore, our primary novel idea is to a-priori restrict the elements of STC updates to lay on such a structured pattern, thus allowing us to further reduce the STC communication cost. This Chapter details the design, implementation, and evaluation of our novel technique, called Structured Sparse Ternary Compression (SSTC). Reported experimental results show that SSTC shrinks compressed updates by a factor of x3 with respect to traditional STC and with a reduction up to x104 with respect to uncompressed FedAvg, at the expense of negligible degradation of the global model accuracy. For the sake of result reproducibilty and to foster related lines of work, we provide a reference to our repository that contains the simulation code we used for experimental results (`https://github.com/alessiomora/SSTC`).

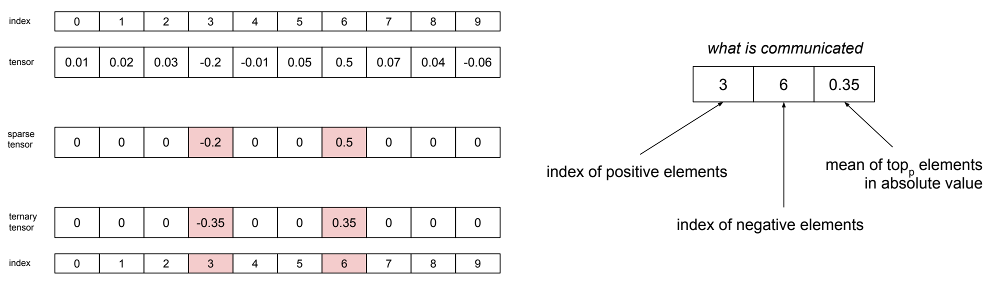


Figure 4.1: An example of STC application to a tensor with 10 elements with p=20%.

4.1 Sparse Ternary Compression

STC [161] is a lossy compression scheme able to extremely reduce the per-round communication cost of FL iterations. STC uses, in series, top_p sparsification and ternary quantization. Firstly, all but the p larger absolute values in the input tensor (either model weights or weight updates) are zeroed out.[1] Then, the survived fraction of elements is binary quantized to $\{\mu, -\mu\}$, with positive values substituted with μ, negative values with $-\mu$, and μ being the mean, in absolute value, of the non-zero elements. Therefore, the algorithm outputs a ternary tensor with values $\{-\mu, 0, \mu\}$.

The ternarization process of STC reduces the entropy of the tensor to be communicated, and favors an efficient encoding. Only the mean value, μ, and the indexes of non-zero elements has to be transmitted, instead of all the values in the original tensor. The decoder assumes that elements corresponding to non-communicated indexes are filled with 0 value. Fig. 4.1 illustrates a simplified example of STC application to a flat tensor with 10 elements.

It is worth noting that indexes can be losslessly compressed by transmitting the distances between consecutive indexes instead of their absolute positions in the tensor. Then distances can be optimally encoded with Golomb code [161], by reducing the bits necessary to represent them[2]. Let us highlight that this can be applied as well in our SSTC proposal; however, we do not consider the possibility of using Golomb encoding for in-

[1] p can be expressed as a percentage of the number of elements in the input tensor. $top_{x\%}$ sparsification retains the $x\%$ larger values.

[2] In [161] the authors assume a random sparsity pattern, and that the distances among consecutive indexes can be approximated by a geometric distribution.

Algorithm 3: FedAvg with clients sending updates Δ instead of complete local models.

```
Server executes:
    initialize $w_0$
    for each round $t = 0, 1, 2, 3, ..$
        $c \leftarrow max(C \times K, 1)$
        $S_t \leftarrow$ (random set of $c$ clients)
        for each client $k \in S_t$ in parallel
            $\Delta_t^k \leftarrow \text{ClientUpdate}(k, w_t)$
        $\Delta_t \leftarrow \sum_{i \in S_t} \frac{n_k}{n} \Delta_t^k$
        $w_{t+1} \leftarrow w_t + \Delta_t$
ClientUpdate$(k, w_t)$
    $w \leftarrow w_t$
    $\mathcal{B} \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
    for each local epoch $e$ from 1 to $E$
        for batch $b \in \mathcal{B}$
            $w \leftarrow w - \eta \nabla \ell(w; b)$
        $\Delta \leftarrow w - w_t$
    return $\Delta$ to server
```

dexes either in STC and SSTC in the comparison that follows in order to better point out the advantages deriving from the only exploitation of the SSTC approach.

4.2 Our SSTC original proposal

As also demonstrated in [193], for convolutional layers, the distribution of client updates in FedAvg is not uniformly distributed. For the sake of clarity, Alg. 3 presents FedAvg with clients sending back to the server model updates instead of full model parameters. A specific pattern has been demonstrated to emerge: the largest (in absolute value) updates form clusters on a subset of kernels. Fig. 4.2 shows the output of top$_{0.1\%}$ sparsification on updates (in absolute value) for a convolutional layer at three different rounds of the federated training. Each column in the depicted heat maps represents a kernel update. The reported results are obtained using the neural architecture and settings described in Sec. 4.3, and refer to the first layer of the network.

On the other hand, in traditional STC the communication payload is almost entirely due to indexes. In our original SSTC we propose to exploit the presence of kernel-wise pattern in the distribution of large updates for convolutional layers to reduce the communication cost of STC. In a practical perspective, the principle is that knowing a priori that the largest elements of the updates will lay on a restricted subset of the convolutional kernels makes possible a more efficient encoding of indexes.

SSTC supposes to find the majority of the largest updates on a subset of the kernels in the convolutional layers, and restricts the search space for the top$_p$ elements by considering only a fixed number of kernels among all the convolutional layers (i.e., the top$_k$ kernels, identified according to the average of the absolute values of the elements in kernels updates – the columns in the heat maps in Fig. 4.2 and Fig. 4.3).

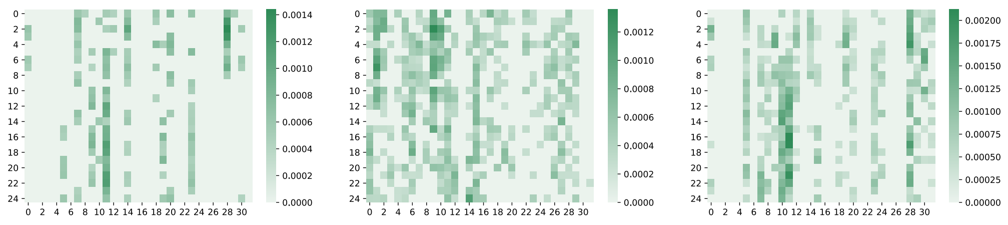


Figure 4.2: Distribution of convolutional weight updates (in absolute value) after $top_{0.1\%}$ sparsification at three different rounds on three randomly sampled clients.

Algorithm 4 formalizes our compression method. For the sake of clarity, Alg. 4 considers a neural network composed by only convolutional layers with the same kernel size. This simplification does not preclude the application of SSTC to neural networks that also have, e.g., fully connected layers, as it is shown also in the reported experimental results.

Lastly, we note that SSTC exactly matches STC when there is no pre-selection of kernels, i.e., when the fraction of kernels to search for larger elements is equal to 1.

Fig. 2 compares the effect of STC and SSTC on the same convolutional weight updates. Different aspects emerge:

- The distribution of the non-zero elements of STC exhibits a column-wise pattern.
- The SSTC approximation of STC is quite accurate even though we are considering, in the depicted results, only the 12.5% of the columns. This is due to the

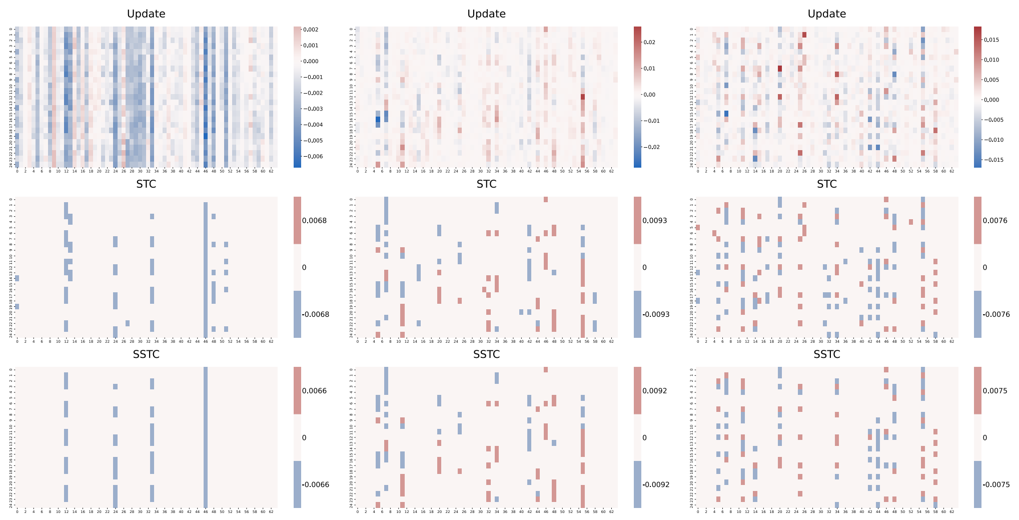


Figure 4.3: The figure reports a comparison among STC-compressed and SSTC-compressed updates at three different rounds (round 1, 500 and 1000). Each column of three heat maps refers to a round. From top to bottom, the heat maps represent the uncompressed updates, the updates after STC application and the updates after SSTC application for that round on a random client. The reported results are obtained using the neural architecture and settings described in Sec. 4.3, and refer to the second layer of the network, with STC sparsity equal to 1% and kernel fraction for SSTC equal to 12.5%. Each column in the heat maps visualizes kernel update.

fact that STC's top_p sparsification zeroes out entire columns while SSTC does not consider such columns in the first place.

- SSTC tends to have more dense kernel updates with respect to STC since it considers a reduced subset of them;
- SSTC inevitably ignores and zeroes out large absolute values that lay on kernels with average lower than the top_k while it may include elements that would have been excluded by STC.

Algorithm 4: SSTC algorithm.
$l \in [1, L]$ with L the number of convolutional layers in the neural network, sparsity p, fraction of kernels k.

```
Input  : list $\Delta$ of per-layer update tensors $\Delta_l \in \mathbb{R}^{K \times K \times C_l \times F_l}$, $p$, $k$
Output: list $\Delta^{sstc}$ of per-layer SSTC update tensors $\Delta_l^{sstc}$
         $\in \{-\mu, 0, \mu\}^{K \times K \times C_l \times F_l}$
1 $\Delta \leftarrow$ Reshaping $\Delta_l$ to 2-d tensors
2 $T \leftarrow$ Concatting reshaped $\Delta_l$ to one 2-d tensor
3 $T^{top_k\_kernels}, indexes \leftarrow$ Selecting $\text{top}_k$ kernels on $T$
4 $T^{top_k\_kernels} \leftarrow$ Flattening $T^{top_k\_kernels}$
5 $T^{stc} \leftarrow$ STC on $T^{top_k\_kernels}$ with sparsity $p$
6 $T^{sstc} \leftarrow$ Scattering $T^{stc}$ column to the original shape of $T$ by means of kernel
    $indexes$
7 $\Delta^{sstc} \leftarrow$ Slicing and reshaping $T^{sstc}$
8 return $\Delta^{sstc}$
```

4.2.1 Lossless Encoding

To transmit the produced structured sparse ternary tensor, the indexes of the top_k kernels and a ternary map of the values' sign in each kernel are communicated. The ternary maps have the size of the kernel. The values in the maps are $\{-1, 0, 1\}$, i.e, 0 for sparsified elements, -1 or 1 to signify the sign of the non-zero elements. The zeros within retained kernels are communicated since we expect the non-zero values to be dense in the selected kernels. Intuitively, the gain stands in indexing groups of consecutive values, instead of indexing values one by one.

Let us note that the above SSTC encoding results to be convenient if compared with STC when it holds that:

$$\frac{b_p * p}{b_k * k + 2 * K * K * k} > 1 \tag{4.1}$$

Where the nominator and denominator in Eq. 4.1 respectively refer to the communication cost due to indexes in STC and SSTC, and $b_k = 1 + \lfloor \log_2(C * F) \rfloor$ is the bitsize to represent each of the kernels in SSTC with F being the number of filters and C the number of channels, $b_p = 1 + \lfloor \log_2(K * K * C * F) \rfloor$ is the bitsize to represent each of the non-zero elements in STC with $K \times K$ being the size of convolutional kernels, k is the number of the communicated kernels in SSTC and p is the number of non-zero values in STC. Eq. 4.1 can be rewritten as:

$$k < p * \frac{b_p}{2 * K * K + b_k} \tag{4.2}$$

It is worth noting that Eq. 4.1 and Eq. 4.2 refer to the application of STC and SSTC on convolutional weights only. Hence, if we consider a neural network with only convolutional layers, $p = sparsity * W$ with W being the total amount of weights. Conversely, if we consider a neural network with, for example, both convolutional and fully connected layers, p will most probably vary round by round, since sparsification is applied considering the whole model parameters, and non-zeroed elements can be distributed among all the layers.

4.3 Experimental Results

4.3.1 Experimental Setup

We consider FedAvg as the baseline and we compare SSTC vs. STC in terms of top-1 accuracy of the global model and of compression ratio. We use the same hyperparameter tuning for every strategy: the local epochs of each client are fixed to 5; Stochastic Gradient Descent (SGD) is used as local optimizer with 0.1 learning rate; the client batch size is fixed to 16. The simulations are run for 1000 rounds with 50 clients (out of 3,400 total learners) randomly selected per round. The sparsity is equal to 0.01 (i.e. 1%) both for

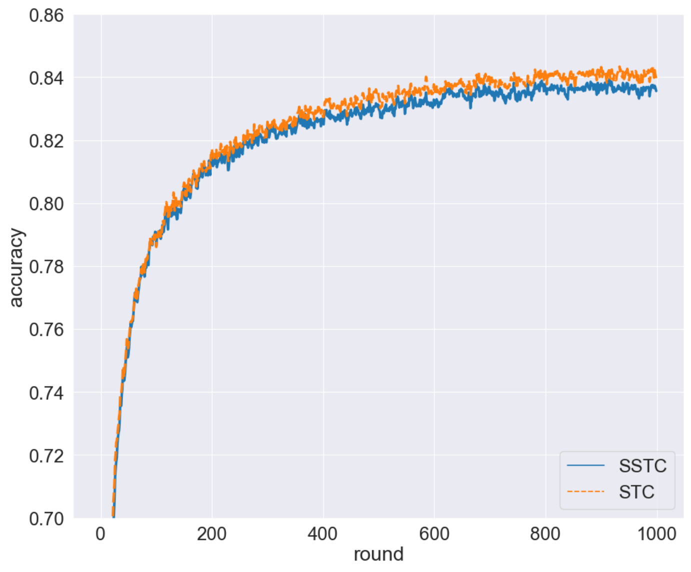


Figure 4.4: Top-1 accuracy of global model on test set for STC and SSTC, with sparsity equal to 0.01 and the fraction of kernel equal to 0.125.

STC and SSTC. We implement the simulation using TensorFlow (TF) and TensorFlow Federated (TFF).

For the experiments, we use the LEAF version of the Federated EMNIST dataset (FEMNIST) available in TFF for 62-class image classification. Each learner holds her/his own handwritten characters to reproduce data heterogeneity, and the labels are unbalanced in number and not uniformly distributed among clients. FEMNIST has a total of 671,585 examples for training, distributed among 3,400 participants, and 77,483 examples for testing. For the FEMNIST classification task, we use a Convolutional Neural Network (CNN) composed by two 5x5 convolution layers, 32 and 64 filters respectively, each followed by a max pooling layer, a fully connected dense layer with 512 units, and a final softmax output layer.

4.3.2 Measured Performance Results and Related Discussion

The first two convolutional layers of the neural network that we considered for the experiments have 52,000 weights in total, excluding biases that are not subject to compression (the first layer has 32 filters with 5x5 kernels size, the second layer has 32 channels with

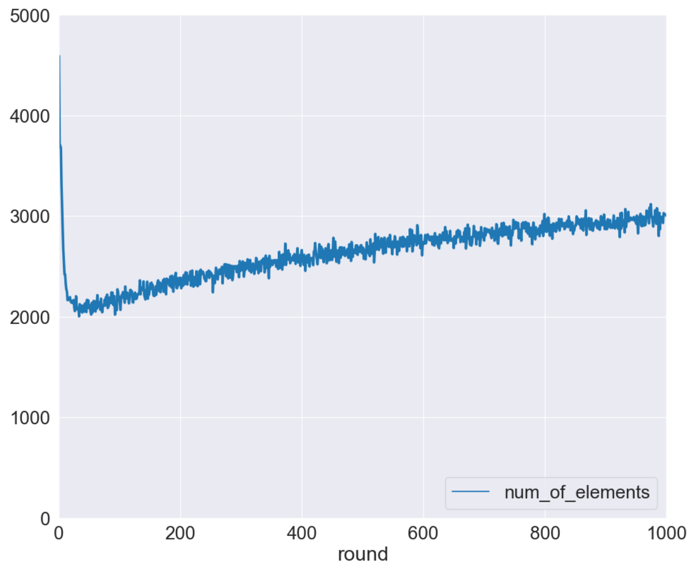


Figure 4.5: Number of elements in STC updates that belong to the convolutional layers during experiments in Fig. 4.4.

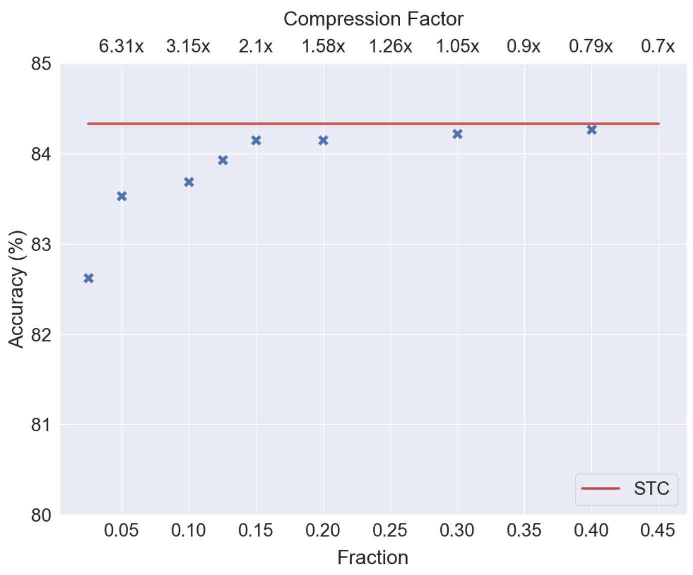


Figure 4.6: Accuracy reached with different tuning of k, expressed as a fraction, in SSTC. The line represents the maximum accuracy of STC. The top x-axis reports the compression factor, averaged within 1000 rounds, of SSTC with respect to STC, i.e. the average gain introduced by the encoding proposed in Sec. 4.2.1 with respect to communicate indexes one by one.

64 filters with 5x5 kernel size). The neural network also has two fully connected layers for classification. As we explained in Alg. 4, the algorithm performs a pre-selection of the kernels with larger element mean, in absolute value, and then consider only such a subset of convolutional weights when sparsification is applied to the whole neural network weights (parameters belonging to both convolutional and fully connected layers).

To compare the compression gain introduced by SSTC with respect to STC, as we noted in Sec. 4.2.1, we monitored how many non-zeroed elements belonging to the convolutional layers are communicated round by round in STC (Fig. 4.5). In this way, we have a range of values for the term p in Eq. 4.1; in the experiments, p ranges among 2,000 and 3,000 (approximately). K is fixed to 5, since kernels have size of 5x5. In STC, to represent the maximum index (i.e., 51,999) 16 bits are needed (i.e., $b_p = 16$), also considering the worst case when encoding using distances. In SSTC the maximum index for kernel depends on the tuning of k. For example, if we select only the 12,5% of the kernels, we have $k = 260$ for the considered neural network, and 11 bits needed to represent them (i.e., $b_k = 11$). So, applying Eq. 4.1, the gain of SSTC with respect to STC considering the convolutional part of the considered neural network ranges, approximately, between 2x and 3x.

The gain introduced by SSTC is more evident if compared with uncompressed FedAvg, as reported in Table 4.1.

Fig. 4.6 depicts the maximum accuracy reached by different options for k tuning. The global model accuracy of SSTC converges to the one of STC when k approaches the 40% of the kernels. However, the proposed encoding is efficient for very dense (i.e., containing few zero values) sign maps, hence with low k value. In fact, in the considered deployment environment, when the fraction of top$_k$ kernels is over (approximately) the 30% the encoding proposed in Sec. 4.2.1 becomes less efficient than directly sending indexes one by one (Fig. 4.6 reports, in the top x-axis, the compression factor of SSTC with respect to STC).

4.4 Related Work

As reviewed in Chapter 3 a plethora of strategies have been recently designed to reduce the per-round cost of Cross-device FL. Sparsification, quantization and encoding, proposed

Table 4.1: Compression factors of STC and SSTC for updates coming from the conv layers with respect to uncompressed (32-bit float) FedAvg. Sparsity = 1%, and k = 12.5% for SSTC.

	compression ($\approx$)	max acc.
FedAvg + STC	41x	84.33%
FedAvg + SSTC (ours)	104x	83.94%

in different flavors represent the main line of work to significantly lower the bandwidth requirements of the decentralised learning process. Also federated pruning [198] and federated dropout [13] result in reduced communication cost for learners, but alone cannot achieve extreme compression as sparsification- and quantization-based compression.

To the best of our knowledge, FedSCR [193] is the only work that analyses and exploits patterns in the distribution of convolutional updates for enhancing compression. They focus on convolutional architecture, as we do in our SSTC. However, the main similarity between our work and FedSCR stands in the attempt of exploiting update patterns for compression scope, while ours and their proposed techniques are deeply different from several other perspectives: in our proposal we try to embed such empirical observations in STC for a more efficient encoding and we act at the kernel level, while in FedSCR they consider channels and filters for their structure-wise identification of more significant update components; furthermore, in FedSCR, learners accumulate updates deemed insignificant, eventually synchronizing them with the server; conversely, SSTC does not assume that the same client will participate more than once in the FL process.

4.5 Concluding Remarks

In this paper we presented an original compression technique that builds on top of STC, and leverages a specific pattern in the distribution of convolutional weight updates to shrink the STC encoding.

While this work may be seen as a specialization of STC, SSTC can be more generally thought as an attempt to exploit – and benefit from – evident patterns in FL updates, a research line that is still largely unexplored in the existing literature and that calls for additional future work by the Federated Learning community. Assumptions on the distribution of client updates can enhance compression techniques, as well as strategies that analyse updates (e.g., identifying malicious updates leveraging autoencoders).

The SSTC technique originally presented here only considers convolutional weight updates; our current research work includes the analysis of the distribution of fully connected or recurrent weight updates in search of meaningful patterns. In addition, it could be interesting to extend the experimental results on deeper CNNs and more complex datasets.

5 Communication-Efficient Heterogeneous Federated Dropout

In this Chapter, we illustrate our novel solution, originally presented in [13], to address device heterogeneity and to extremely reduce the communication cost in Cross-device federated settings. The proposed method is based on the FedAvg algorithm, hence, following the taxonomy (Fig. 7.1) from Chapter 2, we devise a star-shaped synchronous protocol which exchanges model parameters and model updates.

First, we extend the Federated Dropout technique to allow the server to broadcast differently sized sub-models that best fit the capabilities of the learning participants. Then, we drastically reduce the communication cost of the process by applying, in series to Federated Dropout, sparsification-based compression. In addition, we revisit Sparse Ternary Compression to account for single layers of the neural network, and we either use it or traditional STC within our Communication-Efficient Heterogeneous Federated Dropout (CE-HFD). In our experiments, we show that CE-HFD can substantially lighten the computation burden on clients (up to 50% less model weights for local training) and drastically reduce the client-to-server communication payload (up to 191x) without significantly degrading the global model performance.

5.1 Federated Dropout

As reviewed in Chapter 3, in Cross-device FL, stream compression (e.g., by encoding, quantisation and/or sparsification of model weights/updates) is typically employed to reduce the per-round communication cost as widely recognised in literature.

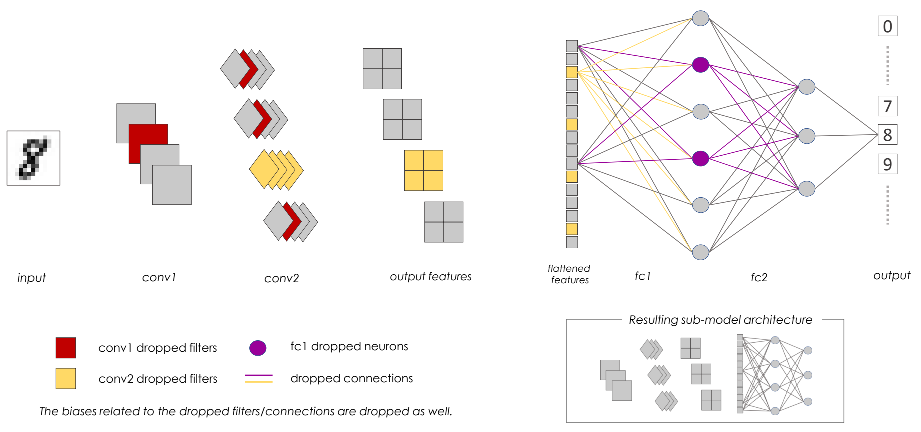


Figure 5.1: How Federated Dropout works for convolutional layers and fully connected layers.

Orthogonally, Federated Dropout [26] has emerged as an elegant solution to conjugate communication-efficiency and computation-reduction on FL clients. Inspired from the well-known Dropout regularization technique [174], Federated Dropout has been proposed to both reducing communication payloads and lightening the computation burden on clients in FL processes. In a nutshell, the server assembles reduced sub-models by randomly dropping out a fixed fraction of the global model neurons and clients train their updates for such smaller sub-models [26]. The fraction of kept neurons is referred to as the federated dropout rate. Therefore, every client receives the same sub-model architecture, but filled with different (random) portions of the global model that, before aggregating the gathered client model updates, have to be mapped back accordingly. A slightly different alternative implementation consists in considering the same dropped elements (i.e., filters for convolutional layers and activations for fully connected layers) for all the clients in a certain round. Fig. 5.1 depicts a simplified example of a Federated Dropout process. The depicted neural network has 2 convolutional layers, i.e. *conv1* and *conv2*, and 2 fully connected layers, i.e. *fc1* and *fc2*, that classifies a character from Federated EMNIST dataset. Max pooling operations are ignored in the figure for the sake of clarity. Note the flattening operation to connect the output of convolutional layers to the fully connected ones. The coloured filters/activations are the ones that will be dropped; the grey filters/activations will be kept in sub-models.

Locally training by using smaller models not only implies having reduced updates in size but also reduced memory footprint for gradients and for the state tensors associated with the local optimizer, e.g., momentum SGD. Furthermore, it is worth highlighting that the forward and backward passes only involve the units survived after the Federated Dropout pruning, hence resulting in reduced computation requirements for clients. In fact, the amount of weights of each fully-connected/convolutional layer is shrunk by a factor of $\frac{N_i \cdot N_{i+1}}{\lfloor d \cdot N_i \rceil \cdot \lfloor d \cdot N_{i+1} \rceil} \approx \frac{1}{d^2}$, with N_i and N_{i+1} representing the number of input/output neurons of a certain layer, and $d \in (0, 1]$ being the dropout rate, and considering the number of surviving neurons/filters rounded to the nearest integer. Bias terms are reduced by a factor of $\frac{N_{i+1}}{\lfloor d \cdot N_{i+1} \rceil} \approx \frac{1}{d}$.

5.2 CE-HFD: Communication-Efficient Heterogeneous Federated Dropout

In order to address a federation of heterogeneous clients with diverse computation/ communication capabilities, we use in series two techniques: (i) Heterogeneous Federated Dropout, and (ii) sparsification-based compression, either traditional STC or per-layer STC. A logic overview of our approach, named Communication-Efficient Heterogeneous Federated Dropout (CE-HFD), is depicted in Fig. 5.2.

A round of CE-HFD proceeds as follows. The server broadcasts to each of the selected clients a compressed, custom sized sub-model by mapping the global model to such a reduced sub-model architecture. Clients decompress the received sub-model, and locally train the received sub-models – note that clients can be fully unaware of the global model architecture. Then, clients produces their updates by subtracting the locally computed weights from the received decompressed ones, and further shrink their updates by applying a specific, sparsification-based compression. Next, they send back to the server the lossy compressed model deltas (i.e., compressed updates). The server gathers the computed model deltas, decompresses them to recover the sub-models, re-maps the reconstructed sub-models to the global model and aggregates all the received updates. A new round can start again. The algorithm of CE-HFD is formalized in Alg. 5.

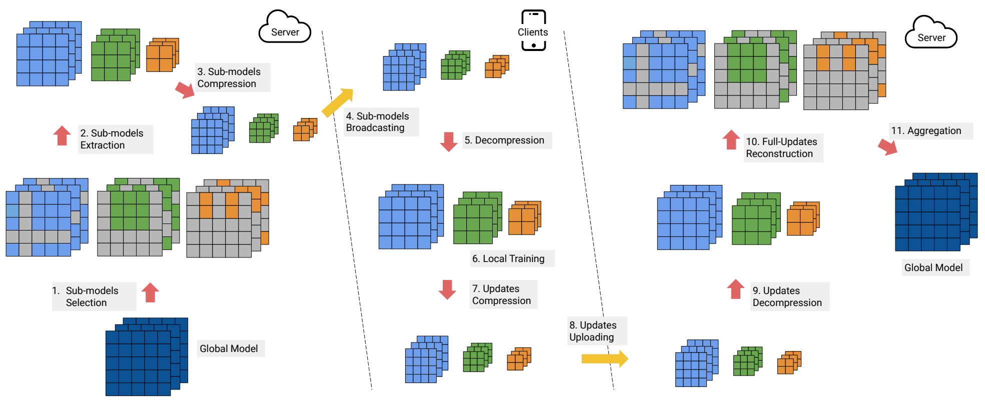


Figure 5.2: The server-side and client-side steps for one round of our CE-HFD strategy.

5.2.1 Heterogeneous Federated Dropout

In known Federated Dropout techniques all the selected clients for a given round receive the same sub-model architecture. We originally extended the idea of Federated Dropout to allow for model heterogeneity considering that the devices that constitute the federation can be clustered in tiers that reflect their capabilities. To this regard, the server maintains a discrete set of predefined sub-models to be then broadcast accordingly to the tier to which a given client belongs. For example, the server may choose among three sub-architectures, each one resulting from a specific tuning of the dropout rate (e.g., 0.8, 0.75 and 0.7). The server assembles the model as in traditional Federated Dropout. To map back the trained sub-models, the server identifies to which sub-model the received weight deltas refer and knows the seed for pseudo-random generation of the dropout mask (i.e., which filters/activation to be dropped), so that it can reconstruct the updates for the larger global model. It is worth noting that in our solution, when mapping the sub-models back to the global model, the server considers the weights of the dropped filters/activations as zeros, and uses the original FedAvg aggregation step (i.e., line 8 of Alg. 1), apart from possibly using other server-side optimizers (not only mini-batch SGD, see also Chapter 7 for details).

Algorithm 5: CE-HFD Algorithm.
tier defines the sub-model to be assigned to a certain client, s the seed for pseudo-random generation of activations/filters has been dropped, the *comp* prefix underlines compressed entities.

```
Server executes:
    initialize $w_0$
    for each round $t = 1, 2, 3, ..$
        $m \leftarrow max(C \times K, 1)$
        $S_t \leftarrow$ (random set of $m$ clients)
        for each client $k \in S_t$ in parallel
            $w_t^k \leftarrow$ ExtractSubModel$(w_t, tier, s_k)$
            $comp\_w_t^k \leftarrow$ Compress$(w_t^k)$
            $comp\_\Delta_t^k, s_k \leftarrow$ ClientUpdate$(comp\_w_t^k, s_k)$
            $\Delta_t^k \leftarrow$ Decompress$(comp\_\Delta_t^k)$
            $\Delta_t^k \leftarrow$ Remap$(\Delta^k, tier, s_k)$
        $\Delta_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \Delta_t^k$
        $w_{t+1} \leftarrow$ ApplyGradients$(\Delta_t)$
Clients execute:
ClientUpdate$(comp\_w, s_k)$
    $w \leftarrow$ Decompress$(comp\_w)$
    $w_{initial} \leftarrow w$
    $\mathcal{B} \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
    for each local epoch $e$ from 1 to $E$
        for batch $b \in \mathcal{B}$
            $w \leftarrow$ ApplyLocalOptimizer$(w, b)$
    $\Delta \leftarrow w - w_{initial}$
    $comp\_\Delta w \leftarrow$ Compress$(\Delta w)$
    return $comp\_\Delta, s_k$ to server
```

5.2.2 Per-layer Sparse Ternary Compression

STC applies $top_{p\%}$ sparsification and calculates μ to be used in the ternarization process, by considering the weight deltas (i.e., updates) irrespectively from the layer they belong to. However, different layers may have weights of different magnitude, and in turn updates of different scales [126]. Hence, the $top_{p\%}$ sparsification may favour updates coming from a specific layer of the neural network and penalize others. We instead propose to evaluate the employing of STC at the layer level. This gives also the possibility to tune the sparsification in a finer way: specific layers can be compressed more or less aggressively with respect to others. In fact, as we will see from the results reported in the following section, some layers may be more sensible to sparsification, either for their specific connectivity (e.g., convolutional), for their size (e.g., number of filters/activations), and for their placement in the network (shallower/deeper layers).

5.3 Experimental Results

5.3.1 Experimental Setup

We evaluate our strategy against already established FL benchmarks, and in particular we use Federated Averaging (FedAvg) [128] as a benchmark. Both for our strategy and for FedAvg, we fix the local epochs of each client to 5; we use SGD Nesterov momentum with 0.9 momentum as local optimizer. We set the local batch size to 10. We use a learning rate of 0.004. In all experiments, local training is performed with full precision, i.e. 32-bit float. Biases are never compressed since their compression results in a negligible communication saving; they are only reduced in size through HFD.

Dataset

For the experiments, we use the LEAF version [25] of the Federated EMNIST dataset (FEMNIST) [38] available in TensorFlow Federated[1] for a 62-class image classification. FEMNIST reproduces the data heterogeneity (non-IIDness) among clients by having each participants holding her/his handwritten characters, and the distribution of per-client labels is also unbalanced. It has 3,400 users with a total of 671,585 examples for training and 77,483 examples for testing. We implemented the simulations with TensorFlow and TensorFlow Federated.

Model

For FEMNIST's image classification task, we use a CNN composed of two 5x5 convolution layers, a fully connected dense layer with 512 units, and a final softmax output layer.

[1] https://www.tensorflow.org/federated

The first convolutional layer has 32 channels, and the second one has 64 channels; each of them followed by 2x2 max-pooling.

Sub-Models

We define three sub-models resulting from applying 0.8, 0.75 and 0.7 dropout rates to the model described in the previous subsection. Table 5.1 reports how dropout rates translate in reduced model size. We assign one out of these three sub-models to each client selected per round by using a discrete uniform distribution; there is no correlation by the amount of local data examples and sub-model assignment.

Table 5.1: Global model and sub-models sizes comparison.

	# param	dropout rate	size reduction (≈)
global model	1,690,046	0.0	
sub-model (i)	1,084,359	0.8	36%
sub-model (ii)	956,894	0.75	43%
sub-model (iii)	837,373	0.7	50%

Table 5.2: Comparison of different client-to-server compression against full-precision FedAvg. p.l. stands for per-layer. Columns l1, l2, l3, l4 represent the four layers of the considered neural network; l1 is the first convolutional layer. In per-layer STC, 8-bit quantization is applied for non-sparsified layers. Last column reports the rounds for 75% accuracy.

	sparsification				compr.	max. acc.
	l1	l2	l3	l4	(≈)	(round)
FedAvg		-			-	85.36% (168)
STC		.01			100x	83.84% (166)
p.l. STC (1)	.01	.01	.01	.01	100x	82.09% (194)
p.l. STC (2)	-	.05	.0075	.05	100x	82.71% (195)
p.l. STC (3)	-	.1	.005	.1	100x	81.98% (193)
p.l. STC (4)	-	-	.01	-	44.9x	84.55% (191)

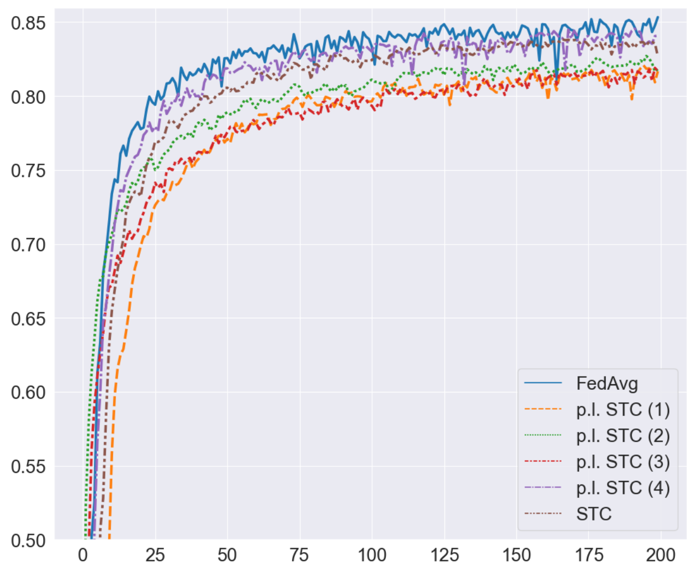


Figure 5.3: Top-1 Accuracy on the y-axis, number of rounds on the x-axis. 50 clients selected per round. Comparison of uncompressed FedAvg, FedAvg plus STC and FedAvg plus per-layer STC in upload.

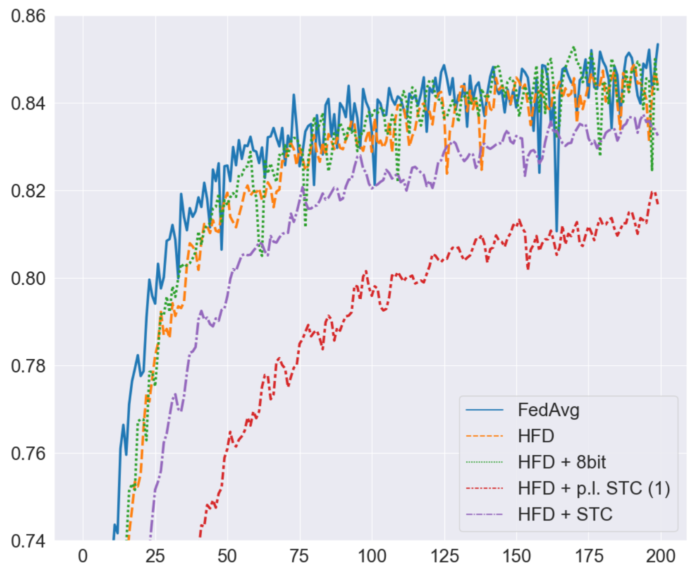


Figure 5.4: Top-1 Accuracy on the y-axis, number of rounds on the x-axis. 50 clients selected per round. Comparison of FedAvg, and CE-HFD with different compression technique and tuning.

Table 5.3: The first row refers to uncompressed HFD, the others entries refer to CE-HFD. Columns s1, s2, s3 refer to each sub-model. Server-to-clients communications are quantized to 8-bit precision.

	sparsification				compr. (≈)			max. acc.
	l1	l2	l3	l4	s1	s2	s3	(round)
HFD		-			1.6x	1.8x	2x	84.89% (191)
HFD 8bit		-			6x	7x	8x	85.29% (170)
STC		.01			148x	168x	191x	83.73% (194)
p.l. STC (1)	.01	.01	.01	.01	148x	168x	191x	81.97% (198)
p.l. STC (4)	-	-	.01	-	66x	74x	83x	84.25% (162)

5.3.2 Results and Analysis

We compared uncompressed FedAvg with per-layer STC and traditional STC both with the same sparsification rate (i.e., 0.01) – that translates[2] to the same compression (≈100x) –, without applying dropout-related techniques. Table 5.2 shows the detailed comparison; Fig. 5.3 depicts the training accuracy over 200 rounds for the different configurations. Traditional STC outperforms per-layer STC; this is probably due to the fact that per-layer STC forces a compression in all the neural-network layers, also in more sensible layers that in STC are less aggressively sparsified. However, per-layer STC can be tuned in a finer way, e.g. by applying less/more aggressive sparsification on specific layer of the neural network. For example, we excluded the first convolutional layer from sparsification, and we applied a sparsification rate of, respectively, 0.05, 0.0075 and 0.05 to the second convolutional layer and to the fully connected layers. That results in a 100x (≈) compression, improving model accuracy with respect to per-layer constant sparsification, but still not reaching the performance of traditional STC. Taking per-layer STC at its most extreme by only applying STC on the larger layer (the first fully-connected layer that retains the ≈ 95% of the model parameters), and by quantize the other weights to 8-bit representation, the model achieves performances comparable to uncompressed FedAvg and still translates to client-to-server communication reduction of 44x.

[2]The compression factor can be further improved by applying lossless compression on the representation of non-zeroed element's indexes.

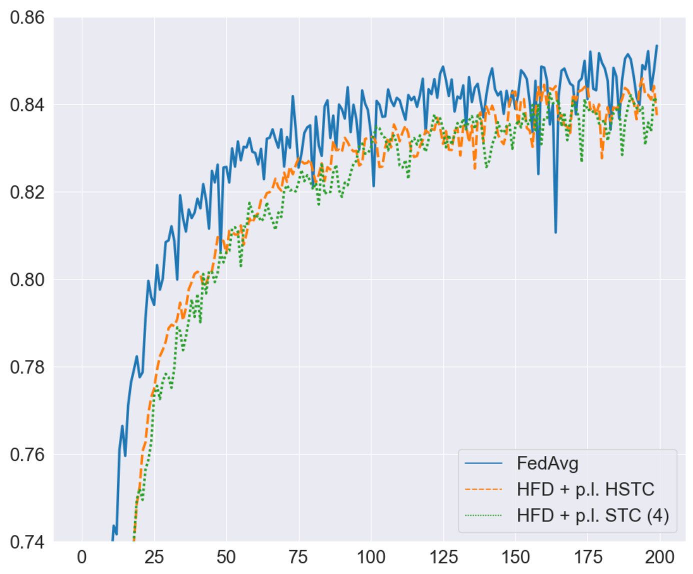


Figure 5.5: Top-1 Accuracy on the y-axis, number of rounds on the x-axis. 50 clients selected per round. CE-HFD with per-layer STC applied only on the larger layer.

Then, we evaluated the performances of HFD, considering sub-models defined as outlined in previous subsection, with traditional FedAvg, both exchanging full-precision (i.e. 32 bit) model weights in broadcast and full-precision updates in upload. We compare these two baselines with different configurations of CE-HFD. Fixing 8-bit quantization as the compression technique for server-to-client communications, we evaluated the following compression techniques for client-to-server updates within CE-HFD: (i) 8-bit quantization, (ii) traditional STC and (iii) per-layer STC with diverse tuning. Table 5.3 details such results. 8-bit HFD shows a faster convergence with respect to its full-precision counterpart while almost reaching the same maximum accuracy of uncompressed FedAvg baseline, even though with a slower start. As happens without HFD, traditional STC outperforms per-layer STC in CE-HFD (see Fig. 5.4). Note that with a sparsification rate of 0.01, the per-layer and traditional STC translate to extreme communication reduction, i.e., 191x update compression for the lower-capacity tier of devices. CE-HFD with traditional STC as compressor almost matches the performance of STC only; HFD does not significantly degrade the model performances in any of the communication-efficient configuration. Furthermore, applying per-layer STC only to the first (the larger) fully connected layer of the neural network, and employing 8-bit quantization for the updates belonging to the rest of the neural network weights, results in model accuracy degraded by slightly more than 1% with respect to the FedAvg baseline, and still translates to 66x, 74x, and 83x update compression respectively for the three considered sub-models.

Finally, we evaluated CE-HFD with different sparsification rate associated to the client tiers. Each client applies a constant 66x compression to the updates with respect to the global model size, irrespectively of the actual sub-model associated with the specific client (that means less aggressive sparsification for smaller sub-models). Such a configuration of CE-HFD is indicated as HFD+p.l.HSTC in Fig. 5.5. Although the latter configuration results in lower overall compression, it does not improve the performance of CE-HFD configured with a fixed sparsification rate for all the three sub-models (see Fig. 5.5).

5.4 Related Work

Bouacida et al. presented a variant of the originally proposed FD, called Adaptive Federated Dropout [21], that maintains an activation score map at server-side to assemble sub-models with filters/activations that show the higher activation score round by round. In their experiments, Deep Gradient Compression (DGC) [116] is applied to further reduce the client-to-server communications. It is worth noting that they use the same sub-model architecture (i.e., same sub-model size) for all the clients. Horvath et al. proposed FjORD [73], a FL framework that exploits Ordered Federated Dropout (OFD), i.e., dropping adjacent components of the model instead of random ones, and introduced the possibility to broadcast differently sized sub-models to meet client computation/communication capabilities as well. Alongside, FjORD uses a self-distillation method to enhance the feature extraction of smaller sub-models resulting in improved global model performances. However, their framework does not focus on further reducing the communications in series to OFD.

Finally, we note that other strategies similar to Federated Dropout in their effect of reducing both computation and communication for clients, but inspired from the Pruning [104] technique rather than Dropout [174], have been proposed in literature, e.g. [198].

Differently from [73], we use the same aggregation step of the traditional FedAvg (see line 8 of Alg. 1) and we employ a random – instead of ordered – Federated Dropout; most importantly, we use sparsification-based client-to-server compression. With respect to [21, 26], we propose to broadcast heterogeneous model architectures, and we use a different compression technique; we also evaluate a variant of the Sparse Ternary Compression (STC) technique proposed in [161].

5.5 Concluding Remarks

In this Chapter, we presented our CE-HFD, which can cope with device heterogeneity and address communication bottlenecks while almost reaching the performances of uncompressed FedAvg. We introduced and evaluated per-layer STC; while classical STC outperforms per-layer STC, the latter allows to finely tune the compression applied to each layer of the neural network. Possible extensions of the work presented in this Chapter include extensions of CE-HFD implementation to other neural-network models and to additional classes of neural networks (e.g., RNNs) as well as using additional federated datasets for evaluation.

6 Knowledge Distillation in Federated Learning

Federated Averaging (FedAvg) represents the baseline algorithm for Federated Learning (FL) [128] with the collaborative learning proceeding in synchronous rounds by leveraging a client-server paradigm. However, parameter-averaging aggregation schemes, such as FedAvg, have well-known limits. Firstly, this class of algorithms implies model homogeneity among the federation, i.e. each client is constrained to use the same neural architecture since the server directly merges clients' updates (e.g., by weighted average). This may be an issue when the federation of learners is composed of clients with heterogeneous hardware capabilities. Furthermore, exchanging model parameters and model updates have high communication cost which scales with the number of model parameters – even though a plethora of strategies (as we reviewed previously in this Thesis) have been proposed to extremely improve the communication efficiency at the cost of global model performance. In addition, exchanging model parameters/updates exposes client to information leakage, and the server must know the architecture and structure of clients' model to broadcast the global parameters, possibly incurring in intellectual property issues (i.e., clients in the federation are unwilling to share the architecture they are using). Lastly, but not less important, when clients hold heterogeneous data, local models tend to diverge from each other during training and fine tune on private examples (i.e., client drift). As a consequence, directly aggregating model parameters/updates degrades global model performances [76, 91, 105, 110, 111, 162, 188, 221].

This Chapter focuses on reviewing federated adaptations of regular Knowledge Distillation (KD) that have been employed to alleviate the above mentioned weaknesses of FL parameter-averaging aggregation schemes. Initially, KD-based strategies, also motivated by encouraging privacy properties [146], have been introduced to enable model heterogeneity and to reduce the communication cost of the process by exchanging model outputs and/or model-agnostic intermediate representations instead of directly transferring model parameters/model updates [29, 36, 56, 64, 77, 81, 83, 103, 160, 195]. Then, a set of strategies have been proposed to enhance the aggregation step of FedAvg with a server-side ensemble distillation phase to enable model heterogeneity and/or improve model fusion in presence of heterogeneous data [30, 115, 159, 213, 214]. Recently, two KD-based lines of work have focused on mitigating the phenomenon of client model drift – which makes averaging-based aggregations inefficient – either using regularization terms in clients' objective functions [67, 68, 94, 100, 132, 203, 225] or leveraging globally learned data-free generator [228].

In this Chapter, we provide a review of the current literature about KD-based approach in FL, with the help of tabular comparisons, following an issue-solutions structure. A discussion on the weaknesses of distillation-based FL algorithms and our vision for future research directions close this Chapter.

6.1 Federated Averaging

Assuming that there are K clients over which the data is partitioned, with each client k holding a private dataset D_k, the goal of FL algorithms is to solve a global objective function, defined as a weighted average over all the clients:

$$\min_{w} f(w) := \sum_{k=1}^{K} \frac{n_k}{n} F_k(w), \tag{6.1}$$

where w represents the parameters of the global model to be trained, n_k represents the cardinality of D_k, and $n = \sum_{k=1}^{K} n_k$ is the total amount of examples held by the participating clients. F_k is a generic local objective function (e.g. cross-entropy loss for a supervised classification task).

Federated Averaging (FedAvg) is considered as the baseline for synchronous FL algorithms that adopt a client-server paradigm. FedAvg proceeds in rounds, and, at each round, performs the following steps in order to approximately solve the optimization problem of Eq. 6.1:

1. The global model is broadcast to a random subset of clients;

2. The activated clients locally fine tune the received model for a certain number of epochs;

3. The clients send back an update for the global model (i.e., the difference among the received and the locally computed model parameters);

4. The server collects the updates, aggregates them by weighted average according to the number of local data points held by clients;

5. The server applies the averaged updates to the current global model.

At step (5), the inverse of the averaged updates can be seen as pseudo-gradient, and the server can apply such pseudo-gradient to the current version of the global model by using an optimizer of choice. For FedAvg, the server-side optimizer is Stochastic Gradient Descent (SGD) with a learning rate equal to 1, which is equivalent to add the averaged updates to the previous-generation global model [155].

As already underlined, parameter-based FL schemes as FedAvg dictate that clients must implement the same model architecture since the server directly broadcasts and aggregates model parameters/updates. In Section 6.3.1, we review FL approaches, based on Knowledge Distillation, to enable model heterogeneity, i.e., each client can implement a model architecture of choice.

As examined in Chapter 3, when clients hold data drawn from different distributions, FedAvg suffers from client drift phenomenon which hampers the convergence of the globally trained model. In Section 6.3.2, we will review the FL approaches based on Knowledge Distillation, which tackle data heterogeneity either rectifying the server model resulting from averaging drifted client models or directly limiting client drifts during local training.

6.2 Knowledge Distillation

Knowledge Distillation (KD) methods have been designed to transfer knowledge from a larger deep neural network, the *teacher*, to a lightweight network, the *student* [23, 71]. In the simplest form of KD, the student model learns by mimicking the (pre-trained) teacher model's outputs on a proxy dataset, also called transfer set. If the transfer set is labeled, the student can be trained using a linear combination of two loss functions,

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_{CE}(q^S, y) + \lambda\mathcal{L}_{KD}(q^S_\tau, q^T_\tau) \tag{6.2}$$

$\mathcal{L}_{CE}$ is the usual cross-entropy loss between the true label y (e.g., hot encoded) and the class probabilities q (i.e., soft targets) predicted by the student neural network. Soft targets q_τ are typically produced by applying a softmax layer to the logits z_i so that $q_\tau(i) = \frac{exp(z_i/\tau)}{\sum_j exp(z_j/\tau)}$, where z_i is the i-th value of logits vector z. The temperature τ controls the softness of the probability distribution. q^S is computed with τ set to 1. λ weights the impact of the two loss terms. $\mathcal{L}_{KD}$ is a general distillation loss that measures the agreement between the student soft targets and the teacher soft targets, e.g. via Kullback-Leibler (KL) divergence. Alternatively, $\mathcal{L}_{KD}$ could directly measure the error between student and teacher logits (e.g., mean squared error). We refer to [58] for taxonomy and recent progress in the KD area.

6.2.1 Codistillation

Codistillation (CD) refers to an online version of distillation, which obviates the need of a pre-trained teacher in regular KD [8, 58]. In fact, codistillation simultaneously trains T copies of a model by adding a distillation term to the regular loss function of the jth model to mimic the average prediction of the other $T-1$ models. In this way, each worker network sees the ensemble of the other models as a virtual teacher.

For CD, the pre-trained teacher's soft targets in Eq. 6.2 is replaced with the ensemble soft targets of $T - 1$ workers.

In the original formulation of codistillation [8], (1) all the workers implement the same neural architecture[1], (2) all the workers use the same dataset for training and, most notably, (3) the distillation loss is employed during training before any model has fully converged. In the following, we classify a set of FL algorithms as *federated adaptations of codistillation* since such strategies train in parallel multiple client models using a distillation loss or adding a distillation term to their loss before any model has converged, and each client sees the ensemble of clients at round $t - 1$ as a virtual teacher. On the other

[1]The requirement of workers implementing the same model architecture is needed to use codistillation to fasten data-center distributed training of large neural networks.

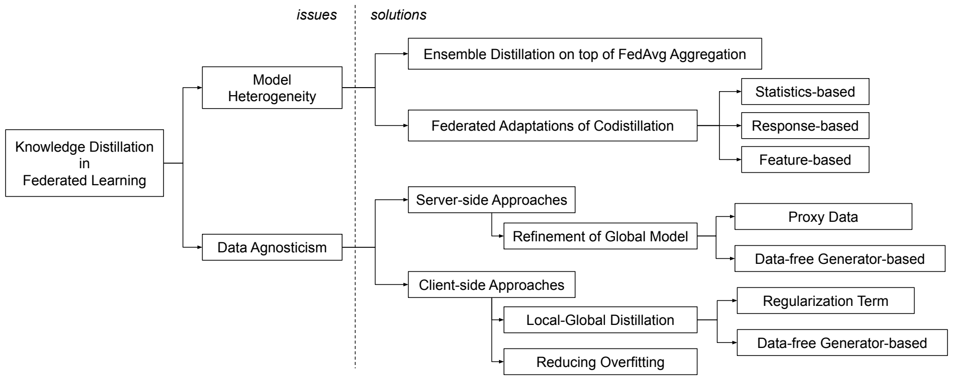

issues
solutions
Knowledge Distillation in Federated Learning
Model Heterogeneity
Data Agnosticism
Ensemble Distillation on top of FedAvg Aggregation
Federated Adaptations of Codistillation
Statistics-based
Response-based
Feature-based
Server-side Approaches
Refinement of Global Model
Proxy Data
Data-free Generator-based
Client-side Approaches
Local-Global Distillation
Regularization Term
Data-free Generator-based
Reducing Overfitting

Figure 6.1: A schematic visualization of KD-based solutions for FL issues.

hand, with respect to regular codistillation, such federated adaptations avoid the need of a shared training dataset and relax the constraint of implementing the same model architecture among clients.

6.3 Knowledge Distillation in Federated Learning

During recent years, KD has been increasingly employed in FL algorithms. In this Chapter, we make a primary distinction according to the purpose KD is used for, identifying two main lines of work: (1) FL algorithms that use KD to enable model heterogeneity (FL model heterogeneity FL); (2) FL algorithms that use KD to mitigate the impact of data heterogeneity on global model performance (data-agnostic FL). Then, we further structure the review according to how these purposes are achieved. For model heterogeneity, we distinguish among (a) solutions that leverage server-side ensemble distillation on top of FedAvg's aggregation phase and (b) communication-efficient strategies that enable model heterogeneity via federated adaptations of codistillation, which we further classify accordingly to the type of exchanged knowledge between clients and server. In regard of strategies to mitigate the degradation introduced by non-IIDness, we differentiate (a) server-side strategies that refine FedAvg's aggregation with a distillation phase and (b) client-side techniques that locally distill global knowledge to directly tackle client drift. If not differently specified in the text, the reviewed solutions adopt a server-client paradigm – the majority in literature – and implement synchronous protocols, which proceed in rounds.

6.3.1 FL Model Heterogeneity via KD

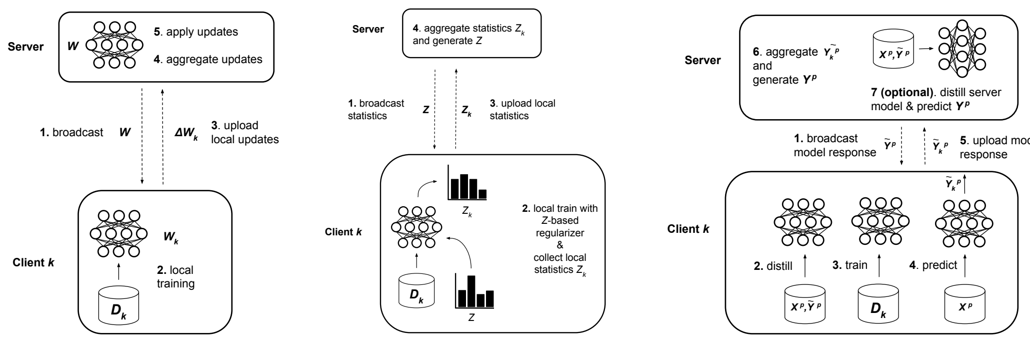
KD has been initially designed to transfer knowledge among neural networks with different structure and depth. In this Subsection, we review strategies that adopt KD to enable model agnosticism in FL, i.e. transfer knowledge among clients with heterogeneous model architectures.

Enhancing FedAvg aggregation

FedAvg's protocol can be enhanced to enable model heterogeneity by leveraging server-side ensemble distillation on top of the aggregation step [115, 159], through which knowledge is transferred among clients with different model architecture. To this end, the server can maintain a set of prototypical models, with each prototype representing all learners with same architecture. After collecting updates from clients, the server firstly performs a per-prototype aggregation and then produces soft targets for each received client model either leveraging unlabeled data or synthetically generated examples. Next, such soft targets are averaged and used to fine tune each aggregated model prototype. Alternative possible solutions to enable model heterogeneity consist in exploiting distributed adaptations of codistillation [8] instead of parameter-averaging algorithms such as FedAvg, as presented in the following.

Federated adaptations of codistillation

The strategies reviewed here can be seen as federated adaptations of codistillation (CD) [8]. In a general federated adaptation of CD, each client at round t acts as student and sees the ensemble of clients knowledge at round $t-1$ as a virtual teacher knowledge. As highlighted in Section 6.2.1, traditional codistillation postulates that the worker networks have access to the same training dataset to form an ensemble of model responses on common samples. Such a requirement is not acceptable in the FL context, where collaborative training is performed without disclosing the private raw data of clients. Therefore, considering a classification task, federated adaptations of CD avoids the aforementioned problem of collecting model responses on common training data examples by exchanging



(a) FedAvg. (b) Statistics-based federated codistillation. (c) Response-based federated codistillation.

Figure 6.2: Visualization of FedAvg baseline (6.2a) with respect to statistics-based federated codistillation (6.2b) and resonse-based federated codistillation (6.2c). Dashed lines indicate communications among parties.

a different kind of knowledge. Among the solutions in the literature, we identify three types of knowledge to enable federated versions of codistillation, and they are:

1. ensemble of aggregated statistics of model responses on local data (e.g. per-label mean model responses),
2. ensemble of local model responses computed on a publicly available dataset and not on local data,
3. or ensemble of both model responses and model-agnostic intermediate features.

It is worth noting that the clients and the server exchange this kind of information instead of model parameters. Furthermore, FL adaptations of codistillation relax the constraint of implementing the same model architecture, which was needed in regular codistillation for datacenter-oriented distributed training. In fact, exchanging knowledge based on model responses (or on model-agnostic intermediate features) enables heterogeneous models among workers, i.e. FL clients, as long as they have the same output shape. Table 6.1 sums up the comparison among FL adaptations of codistillation.

Disclosing aggregated statistics of model responses on local data In [83] Jeong et al. presented a pioneering distillation-based baseline for FL, FedDistill. Participants periodically transmit only per-label mean soft targets computed on their private dataset. The server, in turn, averages such tensors and produces per-label global soft targets to be broadcast the next round. When locally training, clients regularize their local loss with a per-label distillation term which uses the received global soft targets as the teacher's output. A similar strategy is later presented in [165]. It is worth noting that FedDistill is extremely communication efficient with respect to parameter-based schemes when considering DL models for classification task, since the communication payload size depends on the model response size and not on model size.

Exchanging model responses on publicly available data. Federated codistillation can be enabled by using knowledge formed by an ensemble of model responses computed on a proxy transfer set (publicly available, both clients and server can retrieve it). In this way, clients train on their private data, and share knowledge via their model response on the transfer set. Here, the approaches in literature are more variegated, but a general skeleton of algorithmic steps can be the following:

1. *broadcast*: clients receive the current global logits/soft targets;

2. *local digest*: clients distill their local model by mimicking the received global logits/soft-labels on a subset of the transfer dataset. Here the parallel with traditional codistillation: each client sees the averaged predictions of the other clients at the previous round as a virtual teacher. This step can be also seen as a way to retrieve the global model parameters instead of directly receiving them from the server as in parameter-based schemes (e.g., FedAvg);

3. *revisit (local train)*: clients fine-tune the distilled model on local data;

4. *local predict*: clients compute their local logits/soft targets on a subset of the transfer dataset;

5. *upload*: clients sends back the computed logits/soft targets;

6. *aggregate*: the server aggregates the client predictions to produce the updated global logits/soft targets. Next, a new round begins.

While a subset of solutions use the server entity just as aggregator for locally computed model responses [29, 81, 103] (*6. aggregate*), more recent strategies add an additional step to distill a server-side model (*7. server digest*), with the server model being used to produce the global logits/soft targets to broadcast [36, 77, 160]. Learning a server-side model can improve the training process when there is partial participation of clients [160]. Also, considering either a labeled or unlabeled proxy dataset influences the design of algorithms. FedMD [103] uses a proxy labeled dataset to perform an initial pretraining phase on clients, before the protocol starts. Itahara et al. modify the *6. aggregate* step proposing an Entropy Reduction Aggregation (ERA), demonstrating that using a temperature lower than 1 when applying softmax to the aggregated logits reduces the entropy of global soft targets, and can help the training process, especially in non-IID settings [81]. Compressed Federated Distillation (CFD) [160] implements an extreme and effective compression technique for soft targets based on quantization and delta coding, which is applied both by clients and server before communicating. Cronus [29] merges the *2. digest* and *3. revisit* step by directly training on the union (i.e., concatenation) of the private dataset and the soft-labeled public one. In addition, Cronus aggregates (*6. aggregate* step) soft targets following the approach of Diakonikolas et al. [41] for enhanced robustness. Similarly to Cronus, in MATH [77] clients jointly train on private dataset, public dataset, and public dataset tagged with global soft targets. MATH [77] considers a labeled proxy dataset, and distills its server model by training it on the union of such a public dataset with the soft-labeled version of it. FedGEM [36] adopts a protocol that matches FedMD, additionally enhancing it with a server model similarly to CFD. The intuition of FedGEM is to take advantage of a powerful model server. FedGEMS, a variation of FedGEM, exploits the labels in the public transfer set to enforce a selection and weighting strategy which can improve the knowledge transfer [36].

Leveraging intermediate features. FedAD [56] also uses intermediate features besides model output to extend response-based knowledge distillation. The intermediate features are model-agnostic attention maps [121, 164], which still enable model heterogeneity as long as there is consensus on attention map shape. FedAD is a one-shot federated learning framework, which means that clients do not have to distill their local model at the beginning of each round, and can participate asynchronously. FedGKT [64] uses intermediate features tacking advantage of both asynchronous split learning paradigm

[150] and regular FL.[2] Edge devices train small networks composed of a feature extractor, which produces intermediate feature maps, and a classifier, which produces soft targets. Similarly, the server leverages a deeper network and a classifier. After local training, for each local examples, clients communicate their computed intermediate features, the predicted soft targets and the related ground truth labels. The server takes locally computed extracted features as input for its deeper network and produces global soft targets. Both clients and server use a linear combination of regular cross-entropy loss and KD-based loss as objective function. The first considers soft targets and ground truth labels, the latter measures the discrepancy among local and global logits. A similar framework is implemented and extended in FedDKC [195], where Wu et al. also develop server-side knowledge refinement strategies.

Table 6.1: Comparison among strategies to enable model heterogeneity via FL adaptations of codistillation. D_k represents the local private dataset of a generic client. D_p represents a public transfer set. $(X_p, \tilde{Y_p})$ is the public transfer dataset labeled with soft targets $\tilde{Y_p}$. Knowledge column is inspired by the classification in [58]; statistics-based disclose aggregated statistics (e.g., per-label mean logit vector) of client model responses on local data, response-based methods communicate model outputs, feature-based also share intermediate representations.

	Knowledge	Transfer Set	Server Model	Notes
FedDistill [83]	statistics	data-free	no	KD-based regularizer
FedMD [103]	response	labeled	no	Pre-training on D_p
Cronus [29]	response	unlabeled	no	Local training on $(X_p, \tilde{Y_p}) \cup D_k$
DS-FL [81]	response	unlabeled	no	Entropy Reduction Aggregation
MATH [77]	response	labeled	yes	Server training on $(X_p, \tilde{Y_p}) \cup D_p$
CFD [160]	response	unlabeled	yes	Compressed soft targets
FedGEMS [36]	response	labeled	yes	Server-side kowledge refinement
FedAD [56]	feature	unlabeled	yes	One-shot algorithm
FedGKT [64]	feature	data-free	yes	FL + Split learning paradigm
FedDKC [195]	feature	data-free	yes	Knowledge refinement

[2]Split learning refers to a paradigm in which the DL model to be trained is split among server and clients. The server holds the deeper layers of the neural network. During forward pass, activations in output from the split layer are communicated from clients to server along with labels of data samples. Then, the server concludes the forward pass, starts the backward pass and sends back the gradients computed at the split layer so that clients can complete the model update.

6.3.2 Data-distribution-agnostic FL via KD

KD-based solutions can be used to handle data heterogeneity either at server side, rectifying FedAvg's global model via ensemble distillation on a proxy dataset [30, 115, 159] or using a data-free generator [213, 214], or at client side, reducing local overfitting [132] or distilling global knowledge via on-device regularizers [67, 68, 94, 100, 203] or synthetically-generated data [228], directly controlling the phenomenon of client drift.

Server-side KD-based refinement of global model

In [115], the authors propose FedDF, a server-side ensemble distillation approach to both enable model heterogeneity and enhance FedAvg's aggregation. In FedDF, the global model is fine tuned imitating the ensemble (e.g., weighted average) of clients' model output on a proxy dataset. FedAUX [159] boosts the performances of FedDF [115] leveraging unsupervised pre-training on auxiliary data to find a suitable model initialization for client-side feature extractor. In addition, FedAUX weights the ensemble predictions on the proxy data according to (ϵ, δ)-differentially private [48] certainty score of each participant model. FedBE [30] proposes to combine client predictions by means of a Bayesian model ensemble to further improve robustness of the aggregation instead of averaging the model predictions. While server-side ensemble distillation approaches suppose the existence of a proxy dataset, FedFTG [214] performs a server-side refinement of the global model via data-free knowledge distillation where the server adversarially trains both a generator model and the global model, and fine-tunes the latter with synthetic data. A data-free generator-based refinement of global model is also proposed in [213]. It is worth noting that server-side global model rectifications, as the ones reviewed in this Subsection, are orthogonal to client-side approach to control model drift, as the ones presented in Section 6.3.2, and can be used in combination [214].

Client-side KD-based regularization

Local regularization to reduce overfitting In [132], Mendieta et al. show that GradAug [201], a distillation-based structural regularization not specifically designed for FL settings, effectively mitigates client drift issues though substantially introducing computation overhead. Hence, Mendieta et al. design a novel method, FedAlign [132], which has similar effect and performances but with a sustainable computation overhead.

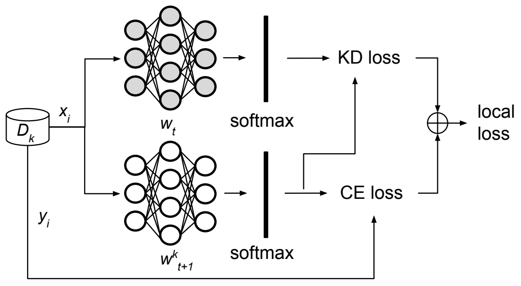


Figure 6.3: Overview of approaches that distill global knowledge using a regularization term during local training. D_k is the private dataset at client k, with x_i and y_i respectively being the data sample i and the corresponding ground-truth label. w_t represents the global model at round t. w^k_{t+1} represents the local model.

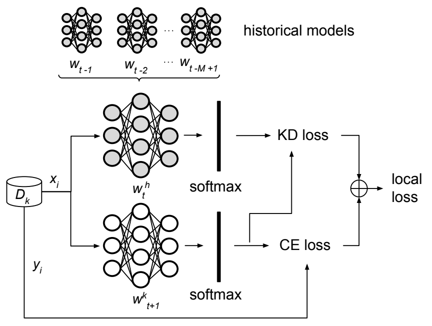
In particular, FedAlign targets the deeper layers of a neural network, most prone to overfit client distribution [124], imposing a distillation-based term in the local objective function. Such term minimizes the discrepancy among the intermediate features produced in output by the final block of the full network and the features produced by the same block but at reduced width (via temporary uniform pruning). The discrepancy is measured via mean squared error of the approximated Lipschitz constants (i.e. top Hessian eigenvalues) of the two intermediate representations. Employing a slimmed sub-block permits to introduce a limited computation overhead.

Local-global distillation via regularization term. Respectively inspired by fine-tuning optimization ideas and continual learning research, the recent works in [100] and [203] find that local KD-based regularization is an effective way of reducing the influence of non-IID data in FL settings. [3] In local-global distillation, the local objective function of clients becomes a linear combination between the cross-entropy loss and a KD-based loss,

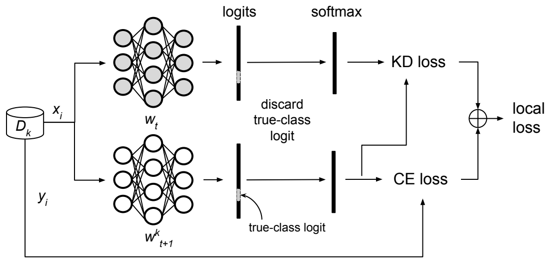
$$\mathcal{L} = \mathcal{L}_{CE}(q^{w^k_{t+1}}, y) + \beta\mathcal{L}_{KL}(q^{w^k_{t+1}}, q^{w_t}) \tag{6.3}$$

where $q^{w^k_{t+1}}$ and q^{w_t} are the soft targets produced on local data respectively by the local model of client k and by the received global model. β weights the impact of the KD-based term. The KD-based loss measures the discrepancy among the global model's output (i.e., the teacher model's output) and the local model's output (i.e., the student model output)

[3] Local-side regularization strategies, as we will detail in the next Chapter, which do not use KD, have been proposed as well, e.g. in [2, 91, 110, 169].



(a) FedGKD.



(b) FedNTD.

Figure 6.4: Overview of FedGKD and FedNTD.

on private data, e.g. via Kullback-Leibler divergence, and works as a regularization term. Fig. 6.3 depicts the basic framework for this kind of local-global distillation.

The inspiration for such a framework is twofold. In [203], Yao et al. borrow the idea from the work in [199], where, in a non-federated setting, self-distillation mechanisms are shown to improve the fine-tuning of pre-trained models such as BERT[40]. In self-distillation, knowledge from past snapshots [200], i.e. produced at previous training steps of the in-training model, assists the current step of model training. Orthogonally, in [100], Lee et al. observe a phenomenon similar to catastrophic forgetting [57] in continual learning research: in presence of heterogeneous data, FedAvg-trained global models exhibit inconsistent predictions on test data between subsequent rounds (i.e., the global model at round $t+1$ shows reduced performance on classes that the global model at round t predicted correctly). Local distillation of global knowledge is shown to mitigate forgetting among subsequent rounds, and in turn to alleviate the harmfulness of data heterogeneity [100].

Local-global distillation via regularization term: further improvements. FedGKD [203] uses an ensemble of M historical global models as teacher for the KD-based regularization where such an ensemble model is computed as the average of M past global models. Fig. 6.4a visualizes how FedGKD works. FedGKD-VOTE is also proposed as a variation that considers the averaged discrepancy of all the M historical models' outputs as the regularization term [203]. In the simplest formulation of FedGKD, i.e. with $M = 1$, the communication cost is the same of FedAvg, while for $M > 1$ the server-client communication cost is doubled, and for FedGKD-VOTE it scales with

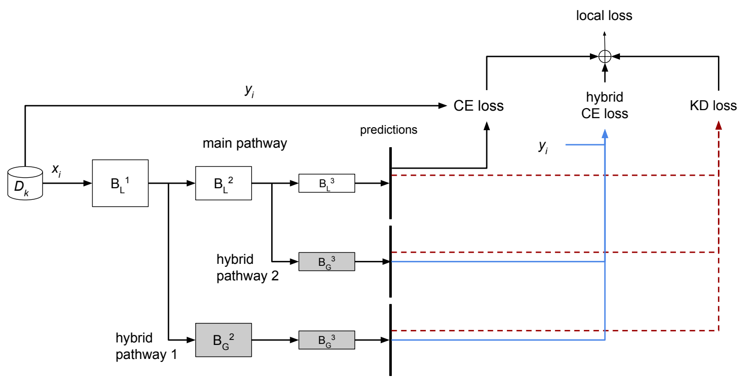


Figure 6.5: Overview of FedMLB. B_L^i and B_G^i respectively indicate local and global blocks (i.e. a series of layers).

M. To reduce forgetting among subsequent rounds of learning, FedNTD [100] ignores the logits produced by the true classes when computing the softmax score later fed to the KD-based loss, as depicted in Fig. 6.4b. Inspired by the work of Lukasik et al. [123], He et al. further observe that, in the framework of Fig. 6.3, leveraging an inaccurate global model (i.e., inaccurate teacher) on specific classification classes might mislead local training [68]. To alleviate such phenomenon, a class-wise adaptive weight is proposed in FedCAD [68] to control the impact of distillation: when the global model is accurate on a certain class, local models learn more from the distilled knowledge. FedCAD determines the class-wise adaptive weight based on the performances of the global model on an auxiliary dataset, and the server broadcasts such information along with model parameters round by round. FedSSD [67] extends FedCAD by also considering the credibility of global model at the instance level when computing the distillation term in local training. FedMLB [94] enhances the local-global distillation also using intermediate representations, preventing them from deviating too significantly during local fine tuning. To this end, FedMLB crafts hybrid pathways composed of local and global subnetworks, i.e. of local network blocks followed by non-trainable global blocks. Besides regular cross-entropy, local learning also considers the average cross-entropy from hybrid paths and the average KL divergence between the outputs produced by the hybrid paths and the main path as regularization term. Due to backpropagation through hybrid pathways, FedMLB locally introduces a moderate computation overhead. FedDistill$^+$, used as alternative baseline in [203, 228], extends the work of [83, 165] by exchanging model parameters in addition to per-label local logits on training dataset. With respect to the framework in Fig. 6.3, FedDistill$^+$ uses the received per-label globally averaged logits – instead of the output of the global model on private data – to calculate the KD loss.

Local-global distillation via data-free generator models. Differently from the other work in this subsection, FedGen [228] learns a lightweight server-side generator which is distributed, round by round, to clients that sample it to obtain augmented training examples, using global knowledge as inductive biases in local learning. To build the generator, FedGen needs to disclose local model parameters (at least the classifier weights) and local label count.

6.4 Comparison of Existing Solutions, Adoption Guidelines and Future Directions

Table 7.2 lists the solutions reviewed in this Chapter, by classifying them according to their primary aim, and by detailing the kind of per-round exchanged information, the need of auxiliary data, and the type of KD involved. In short, the main take away from Table 7.2 is that KD-based FL solutions can enhance collaborative learning under some perspectives while introducing other trade-offs to consider for an appropriate selection and adoption.

6.4.1 FL model heterogeneity via KD

Federated adaptations of codistillation can enable model heterogeneity, and can reduce the communication requirements at the cost of computation overhead with respect to parameter-based schemes. Hence, despite being extremely communication efficient, it may be not always possible to deploy such algorithms on resource-constrained devices due to the overhead of client-side distillation (in Table 7.2, solutions which use *digestion* at client side), while being a suitable model-agnostic alternative for cross-silo settings.[4] Furthermore, this class of solutions usually performs worse than FedAvg-based baselines

[4] For example, in [160] 80000 data points from a public dataset are used to distill on-device model before local training.

Table 6.2: Concise overview of the surveyed solutions. We have identified 5 possible categories for the primary purpose of the proposed solution, i.e., communication efficiency (CE), model heterogeneity (MH), non-iidness (NIID), server-side aggregation (A), and client drift (CD). Upload refers to the client-to-server link. Symbols: w model parameters, z logit vectors (model output before softmax), $\tilde{Y}$ soft targets (model output after softmax), w^h historical model parameters, Z per-label average logit vectors, Y labels of local data, H intermediate feature maps, A attention maps, α_y per-class adaptive weights, C credibility matrix, c local label count. For the last column, a regularizer-based approach uses KD to regularize local training, generator-based leverages a generator model, digestion means that knowledge is absorbed by imitating teacher outputs on common data.

	Purpose	Exchanged information		Auxiliary data	KD approach	
		Upload	Download		Client-side	Server-side
FedDistill [83]	CE, MH	Z	Z	data-free	regularizer	-
FedMD [103]	CE, MH	$\tilde{Y}$	$\tilde{Y}$	labeled	digestion	-
Cronus [29]	CE, MH, A	$\tilde{Y}$	$\tilde{Y}$	unlabeled	digestion	-
DS-FL[81]	CE, MH	$\tilde{Y}$	$\tilde{Y}$	unlabeled	digestion	-
MATH [77]	CE, MH	$\tilde{Y}$	$\tilde{Y}$	labeled	digestion	digestion
CFD [160]	CE, MH	$\tilde{Y}$	$\tilde{Y}$	unlabeled	digestion	digestion
FedGEMS [36]	CE, MH	$\tilde{Y}$	$\tilde{Y}$	labeled	digestion	digestion
FedAD [56]	CE, MH	z, A	-	unlabeled	-	digestion
FedGKT [64]	CE, MH	z, H, Y	z	data-free	regularizer	regularizer
FedDKC [195]	CE, MH	z, H, Y	z	data-free	regularizer	regularizer
FedDF [115]	MH, NIID, A	w	w	unlabeled	-	digestion
FedAUX [159]	MH, NIID, A	w	w	unlabeled	-	-
FedBE [30]	NIID, A	w	w	unlabeled	-	-
FedFTG [214]	NIID, A	w, c	w	data-free	-	generator
FedZKT [213]	NIID, A	w	w	data-free	-	generator
FedGKD [203]	NIID, CD	w	w, w^h	data-free	regularizer	-
FedNTD [100]	NIID, CD	w	w	data-free	regularizer	-
FedCAD [68]	NIID, CD	w	w, α_y	labeled	regularizer	-
FedSSD [67]	NIID, CD	w	w, C	labeled	regularizer	-
FedMLB[94]	NIID, CD	w	w	data-free	regularizer	-
FedAlign[132]	NIID, CD	w	w	data-free	regularizer	-
FedDistill$^+$ [203]	NIID, CD	w, Z	w, Z	data-free	regularizer	-
FedGen [228]	NIID, CD	w, c	w	data-free	generator	-

(in terms of global model accuracy) [159] – even though they typically improve the performance of non-collaborative training [81]. Moreover, most works in this category suppose the existence of a semantically-similar proxy dataset (in some cases even labeled), which may be an unrealistic assumption in some deployment scenarios and use cases (e.g., for specific medical applications). The pioneering communication-efficient data-free strategy in [83] does not incur in local computation overhead, but it is far from achieving global model test accuracy comparable to FedAvg, as demonstrated in [228], also disclosing possible privacy-sensitive information about private data (i.e., per-label model outputs). Solutions as [64, 195] enable model heterogeneity, are usually more communication efficient than FedAvg, and include resource-constrained devices in the federation, by adopting a split-learning paradigm and by taking advantage of KD-based regularization. However, as shown in Table 7.2, due to their split-learning approach, the solutions in [64, 195] disclose local ground-truth labels, which again may incur in privacy violation.

Future Directions While some seminal efforts are recently emerging [3], model-agnostic KD-based strategies for collaborative learning are still poorly understood theoretically. This is going to call for deeper analysis of convergence properties, as it occurred for parameter-based schemes in the recent literature [188].

6.4.2 Data-agnostic FL via KD

For what relates to the solutions to tackle non-IIDness, KD-based server-side refinement strategies such as [30, 115, 159] can improve FedAvg global model performance in presence of highly heterogeneous data when semantically-similar unlabeled proxy data are available. It is worth noting that this class of algorithms exhibits most improvements when several local epochs are performed between communication rounds and client models tend to drift apart. Also, such algorithms do not introduce computation or communication overhead on clients. Data-free generator models can also be used to perform server-side global model corrections as in [214] or to limit client drift directly at the participating devices as in [228], in both cases at the cost of disclosing local label count.

No additional information has to be disclosed from clients and not even proxy data are needed in solutions that regularize local training by employing global model output on local data, as in FedGKD [203] and FedNTD [100]. In addition, this set of strategies do not introduce significant on-device computation overhead and has the same communication requirements as FedAvg.[5] Although FedGKD and FedNTD – and similar approaches – would require to store two full-size models in memory (the local model as usual and the global model as reference), such limitation can, in practice, be avoided by firstly computing the predictions of the received global model (or historical model) on local data, and then proceeding with local training by overwriting the global model. If limited la-

[5] If FedGKD [203] only considers the current global model as its historical model.

beled proxy data are available, local-global knowledge distillation can be improved as in [67, 68]. When moderate computing overhead is sustainable, local global distillation can be enhanced by using intermediate features and hybrid pathways as in [94], thus significantly improving the effectiveness of local-global distillation. It is worth noting that in FedMLB most of the global model parameters must be stored locally during training, and this cannot be avoided as for regularization based on model responses. As highlighted in [94], client-side regularization can be coupled with standard server-side strategies to boost performances (e.g., FedAdam, FedYogi, FedAdagrad [155], or FedAvgM [76]).

Future Directions Cross-device FL algorithms need to take into account the different hardware capabilities of clients, as resource-constrained devices may struggle or be excluded when the computation, memory, or communication requirements are too demanding. Client-side KD-based algorithms often introduce non-negligible computation and memory overhead, which may prevent their use on these devices. Therefore, there is a growing focus on methods to minimize memory and computation overhead (with respect to FedAvg) on clients (e.g., [132]). On the other hand, server-side KD-based solutions often rely on model generators to overcome the need for semantically-similar datasets, but this comes at the cost of collecting possible privacy-sensitive local information; this may raise privacy-related issues, calling for methods which do not need to disclose additional information about client local data. Additionally, while KD-based solutions are commonly used to enhance the generalization ability of the global model, the impact of these solutions on personalization, which has received growing interest in the community [178], is yet marginally explored with some first contributions (e.g., [86]).

6.5 Concluding Remarks

While distributed adaptations of codistillation have been initially introduced as a mean for both reducing the communication cost of FedAvg-like algorithms and enabling model heterogeneity, KD has been recently explored to tackle non-IIDness, either rectifying the aggregation phase of FedAvg or directly limiting the client drift. This Chapter reviews and compares state-of-the-art KD-based techniques for FL, by classifying them according to their purpose and the way to achieve it. We believe that the presented comparison can provide researchers and practitioners in the field with a practical and useful guide to the primary pros/cons of existing solutions, as well as with practical guidelines for the selection of the most appropriate technique depending on the application case and for the identification of still open research challenges for the near future.

7 Tackling Data Heterogeneity

This chapter provides a detailed overview of the state-of-the-art solutions designed to cope with data heterogeneity. While a parallel line of work focuses on improving local personalization performances in presence of non-IID data [144, 178], this Chapter exclusively considers approaches to enhance the generalization ability of the global model. [1] The considered solutions, unless differently specified, adopt a star-shaped topology (client-server as in FedAvg) and proceed in synchronous rounds; usually, classification tasks are considered for design and evaluation.

For the sake of clarity, as shown in Fig. 7.1, we primarily differentiate the reviewed solutions among client-side optimizations and server-side optimizations. The first class of algorithms modify the local training routine of clients – with respect to FedAvg – to explicitly limit the client drift phenomenon (see Section 7.1). At the other end of the spectrum, server-side optimizations modify the aggregation step of FedAvg to limit the degradation introduced by averaging drifted client models (see Section 7.2). Usually, client-side and server-side optimizations are orthogonal and can be used in combination to amplify the performance gain (e.g., to reduce the total rounds needed to reach global model convergence or to achieve better global model generalization).

In the second part of this Chapter, we provide the researchers and practitioners in the field with the original technical contribution of empirically evaluating and comparing a selection of algorithms, among the ones presented here; the comparative evaluation is the basis for a technical and practical discussion on the possible advantages and weaknesses of the considered solutions when implemented and deployed in real application cases and deployment environments. Furthermore, our code is publicly available for result reproducibility and to foster research advances.

[1] The generalization ability of the global model is measured via metrics, such as loss and accuracy, on a test set. The test set contains unseen examples uniformly distributed among classes.

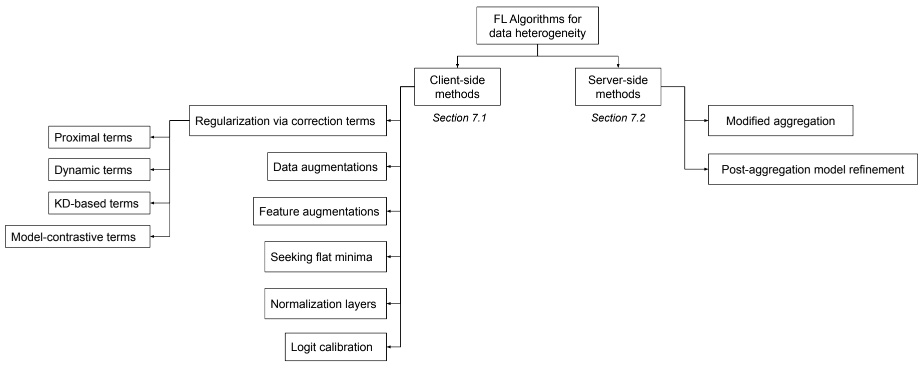


Figure 7.1: Schematic classification of approaches proposed in literature to enhance global model generalization in presence of heterogeneous data.

7.1 Client-side Optimizations

This section goes through solutions that intervene in the local training phase so to limit the degradation introduced by client model drift (see Fig. 3.3) on the global model performance since local models tend to diverge from the received global model and from the model of other clients in the federation in presence of heterogeneous data.

In their pioneering study [221], the authors show via an experimental analysis that FedAvg's test accuracy can be considerably improved in the presence of heterogeneous data by supplementing participants' private datasets with a small portion of the globally shared data provided by the server. While this approach is effective, it comes at the expense of reduced decentralization and necessitates transmitting the common data to the learners.

Without resorting to data-sharing techniques, a common approach to limit the client drift phenomenon is to locally regularize the training phase by introducing one or more additional terms to the client's objective function. This typically translates into applying a penalty that scales with some measure of the client drift, controlling the divergence from the current global model. As discussed in the following, most of the works leverage the current global model as a reference for local training, using it as an anchor for not letting the local model parameters drift apart (e.g., [110]) or using it as a guide during local training (e.g., [67, 68, 94, 203]), or again employing it to prevent *catastrophic forgetting* [100]. Other lines of work do not actively leverage the global model in their solutions, for example focusing on normalization layers (e.g., [112]), or seeking flatter minima in local objective functions [24, 85, 152], or again proposing federated adaptations of traditional

Table 7.1: Classification of reviewed client-side methods, listed by year (from oldest to most recent).

Method	Control Term(s)	Normalization Methods	Logit Calibration	Flat Minima	Feature Aug.	Data Aug.
FedProx [110]	✓					
SCAFFOLD [91]	✓					
FedDANE [109]	✓					
FedMix [204]						✓
SiloBN [7]		✓				
MOON [106]	✓					
FedDyn [2]	✓					
FedBN [112]		✓				
FedRS [113]			✓			
AdaBest [186]	✓					
FedGen [228]						✓
FedGKD [203]	✓					
FedNTD [100]	✓					
FedMAX[33]	✓					
FedCAD [68]	✓					
DMFL [154]			✓			
FedSSD [67]	✓					
FedMLB [94]	✓					
FedAlign [132]	✓					
HarmoFL [85]				✓		
FedLC [211]			✓			
FedSAM [24]				✓		
FedSAM [152]				✓		
FedASAM [24]				✓		
MoFedSAM [152]				✓		
FedFA [224]					✓	
FixBN [222]		✓				
FedTAN [191]		✓				
WSM [24]			✓			

data augmentation techniques [132, 204], or modifying the output *softmax* layer and/or the cross-entropy loss before the network output (e.g., [113, 211]). Authors of [155] also

Table 7.2: Classification of reviewed server-side methods, listed by year (from oldest to most recent).

	Modified Aggregation	Post-Aggregation Refinement
FedAvgM [76]	✓	
FedNova [189]	✓	
FedBE [30]		✓
FedDF [115]		✓
FedOpt [155]	✓	
FedAux [159]		✓
FedZKT [213]		✓
FedDNA [46]	✓	
GMA [181]	✓	
SWA [24]	✓	
FedFTG [214]		✓

provide theoretical convergence analysis, and observe the need for a decaying learning rate at client-side in non-IID settings.

7.1.1 Local Regularizations via Correction Terms

Specifically, FedProx [110] adds a proximal term to the local objective function of clients that is proportional to the *L2-norm* distance between the received global model parameters and the in-learning local parameters, i.e.:

$$\mathcal{L}_{Prox}(w, w_t) = \mathcal{L}(w) + \frac{\mu}{2}||w - w_t||^2 \tag{7.1}$$

where $\mathcal{L}$ is the classic supervised loss for the classification task, μ weights the impact of the L2-norm regularization term, w and w_t are respectively the in-learning local model and the current global model. Besides reducing the impact of heterogeneous data, FedProx was also designed to enable uneven amounts of local training iterations among clients.

Similarly to FedProx, FedDyn [2] proposes a dynamic tuning that is adapted based on the current global model. However, while FedProx does not result in aligning local and global stationary points, during each round, FedDyn dynamically tailors the local objec-

Algorithm 6: FedDyn.

```
Server executes:
    initialize $w_1$
    $\nabla\mathcal{L}(w_0^k) \leftarrow 0$
    for each round $t = 1, 2, 3, ..$
        $c \leftarrow max(C \times K, 1)$
        $S_t \leftarrow$ (random set of $c$ clients)
        for each client $k \in S_t$ in parallel
            $w_t^k \leftarrow w_t$
            $w_t^k \leftarrow \underset{w}{\arg\min}$ Eq. 7.2
            $\nabla\mathcal{L}(w_t^k) \leftarrow \nabla\mathcal{L}(w_{t-1}^k) - \alpha_{dyn}(w_t^k - w_t)$
        // FedDyn needs a server-side state $h_t$
        $h_t \leftarrow h_{t-1} - \alpha_{dyn}\frac{1}{K}(\sum_{k \in S_t} w_t^k - w_t)$
        $w_{t+1} \leftarrow (\frac{1}{|S_t|}\sum_{k \in S_t} w_t^k) - \frac{1}{\alpha_{dyn}}h_t$
```

tive function with a penalty term so that the local optima is asymptotically consistent with stationary points of the global optima. More concretely, the penalty term in FedDyn is composed of a linear term and a quadratic penalty term, and the resulting objective function is in the form of:

$$\mathcal{L}_{Dyn}(w, w_t, w_{t-1}^k) = \mathcal{L}(w) - \langle \nabla\mathcal{L}(w_{t-1}^k), w \rangle + \frac{\alpha_{dyn}}{2}||w - w_t||^2 \tag{7.2}$$

where $\nabla\mathcal{L}(w_t^k) = \nabla\mathcal{L}(w_{t-1}^k) - \alpha_{dyn}(w_t^k - w_t)$ and it is computed recursively, α is a hyperparameter, and clients are indexed by k. [2] It is worth noting that FedDyn is treated here as a client-side optimization, but it also requires a server-side state, and can be used in combination with other client-side optimizations (e.g., FedDyn plus local multi-branch regularization, FedMLB, as presented in the experimental results of [94]). Algorithm 6 displays the FedDyn method. While rooted in theory, FedDyn requires clients to maintain state (i.e., h_t in Algorithm 6) throughout the process. Furthermore, as demonstrated in [186], FedDyn needs a high rate of client re-sampling to exhibit a stable convergence, otherwise, the unbounded increase of h_t norm will cause instability in the learning process.

[2] $\nabla\mathcal{L}(w_0^k)$ is initialized to 0.

Conceptually similar, SCAFFOLD [91] leverages correction terms (i.e., *control variates*) for local gradients to ensure that the local update moves towards the true optimum. Such correction terms are used as proxy for the clients' true gradients, whose disclosure would require unsustainable frequency of synchronization (after each local step, clients should communicate local gradients). Instead, during each FL round, control variates are locally computed and updated, then communicated back to the server for aggregation at the end of local training, and finally aggregated to produce global control variates which are broadcast during the next round. Hence, SCAFFOLD doubles the communication cost with respect to FedAvg.

In addition, it is worth noting that SCAFFOLD and FedDyn require prior knowledge of the number of total clients in order to properly weight their accumulator (i.e., h_t in Algorithm 6), which may be an unrealistic assumption in large-scale FL settings. The authors of [186] propose Adabest, which can be seen as a generalization of FedDyn that does not assume prior knowledge about the number of participating clients and does not require a high re-sampling rate of clients to ensure stability. As a result, AdaBest is most beneficial in the presence of large-scale settings with low participation rates.

FedDANE [109] is a method inspired by DANE [167] and its inexact variant [156], which combines the proximal term used in FedProx with a gradient correction term similar to SCAFFOLD. The update process involves two steps: compute the gradient correction term and solve the Newton-type sub-problem inexactly, the locally computed gradients of the local objective functions are collected and then averaged to approximate the full gradients. To keep from gathering all the locally-computed gradients, FedDANE approximates the complete gradients by aggregating the gradients of a random subset of learners. It is important to note that FedDANE doubles the communication cost with respect to FedAvg, as previously noted for SCAFFOLD. Furthermore, despite being rooted in theory, FedDANE exhibits inferior empirical performance compared to FedAvg and FedProx.

FedMAX [33] focuses on limiting the divergence in *activation vectors*, i.e. input to the neural network's classification layer, among clients and aims at making the activation vectors of same classes as similar as possible in the federation. FedMAX employs a correction term in the loss function of clients to maximize the entropy of such activation vectors, so to limit their uncontrolled divergence.

7.1.2 Local-Global Knowledge Distillation

Besides proximal or dynamic regularizations, also regularization techniques based on Knowledge Distillation (KD) [71] can be employed to tackle the client drift phenomenon by locally distilling global knowledge. Instead of directly considering global and local model parameters as in FedProx, the global model is seen as a teacher in a distillation framework [137], and its model responses on private examples are used to guide the local learning and limit the divergence of client models [67, 68, 94, 100, 203].

FedGKD [203] and FedNTD [100] regulate the local objective function of clients with a term that measures the discrepancy among the output of the global and local models on the same local data. However, such a mechanism does not necessarily transfer the ability of the global model to produce better intermediate features with respect to client models. FedMLB, presented in [94] extends the KD-based framework presented in FedGKD and FedNTD by considering hybrid paths composed of client model blocks followed by global model blocks, encouraging the local model to reproduce intermediate features similar to the ones generated by the global model. The model block is intended as an ensemble of consecutive layers, e.g. a residual block in a residual architecture. However, while FedGKD and FedNTD introduce a negligible computation overhead on clients, FedMLB needs to propagate gradients through the hybrid paths, introducing significant computation overhead and increased local training time. Also, FedGKD, FedNTD and FedMLB require the global model to be stored (and not overwritten as it happens in FedAvg) during local training, hence doubling the memory requirements. This limitation can, in practice, be avoided for FedGKD and FedNTD by firstly computing the predictions of the received global models, and then proceeding with local training, and overwriting the global model.

Considering the local-global distillation framework, the authors in [68] observe that imitating the output of an inaccurate global model as a teacher can misguide local training (for example, in early rounds the global model exhibits poor predictions). To address this issue, FedCAD [68] introduces a class-wise adaptive weight to control the impact of distillation, so that the global model knowledge is distilled when accurate. FedCAD computes the class-wise adaptive weight based on the performance of the global model on a proxy dataset and broadcasts this matrix along with model parameters at each communication round. FedSSD [67] builds on FedCAD by incorporating the credibility of the global model at the instance level when computing the distillation term in local training.

7.1.3 Model-contrastive Learning

MOON [106], inspired by typical data-level contrastive learning frameworks such as SimCLR [32], is a model-level contrastive learning method designed to rectify the local training of clients. While typical contrastive learning compares representations of different augmented views computed on the same image (*positive pairs*) and representations of different augmented versions computed on different images (*negative pairs*), in MOON [106], positive and negative pairs come from the representations computed by the global model and the previous-version local model on the same private example, respectively. The key idea is to correct the local training of individual clients by maximizing the consensus on the representation learned by the current client model and the representation learned by the server model. At the same time, MOON encourages the representation learned by the current client model to diverge from the representation learned by the previous-version client model (client model resulting from the last active round). [3] In practice, this translates to a local objective function composed of a typical term to account for supervised loss (e.g., cross-entropy loss among ground-truth labels and predicted soft labels) and a model-contrastive term. It is worth noting that, differently from FedAvg, MOON implies stateful clients, demanding to store the last trained local model. Additionally, the approach requires the modification of the locally learned model by adding a projection head, also adding computation and storage overhead attributed to the need of producing and comparing the representation of three models (global, current, and last-version local model).

7.1.4 Data and Feature Augmentation

From another perspective, FedMix [204] adapts the simple, well-known $Mixup$ augmentation [210] to the federated context. Similarly, the authors of [132] recently demonstrated that augmentations like GradAug [201] and StochDepth [78] significantly improve the performance of FedAvg in presence of heterogeneous data, though they in-

[3] To produce the representation, MOON [106] injects a projection head before the output layer of the model. The considered representation is the output of the projection head.

troduce substantially computation overhead. Addressing the issue, the authors of [132] design a method, FedAlign, which has similar effects and performance as GradAug in FL but with a reduced computation overhead. In particular, FedAlign aligns the Lipschitz constants (i.e. top Hessian eigenvalues) of deeper layers of the neural network – most prone to overfit client distribution [124] – through the use of slimmed sub-blocks and a distillation-based regularization term. FedGen [228] employs a generator model, broadcast by the server to clients round by round, to be used to locally obtain augmented training examples, while incorporating global knowledge as inductive biases. To train the generator, FedGen requires the disclosure of the local label count, besides local model parameters. While approaches such as FedMix and FedGen works at the input level, FedFA [224] proposes to augment features by injecting Gaussian noise calibrated according to the channel-wise statistics (i.e., mean and standard deviation) of the participants in the federation.[4] In this way, FedFA allows client models to be trained over data examples drawn from more variegated feature distributions, encouraging participant-invariant representation learning, and eventually producing a server model that generalizes better.

7.1.5 Seeking Flat Minima

Recently, the technique of Sharpness-Aware Minimization (SAM) [51] has been employed and proposed in FedSAM [24, 152]. In fact, the authors of [24] suggest that the geometry of the loss surface is a relevant indicator for generalization, and in particular that sharp minima translate to a lack of generalization. To this end, FedSAM locally encourages flatter minima and smoother loss landscape by leveraging SAM, and resulting in improved performance of global model. Also, adaptive and momentum versions of FedSAM have been proposed to speed up convergence, respectively named FedASAM [24], based on Adapative SAM [98], and MoFedSAM [152], which leverages a momentum parameter.

Similarly inspired, HarmoFL [85] promotes client models that are easy to aggregate via weight perturbation (clients that exhibit flatter loss landscape are more amenable to be aggregated via FedAvg's averaging scheme). In addition, HarmoFL couples the weight-perturbation strategy with amplitude normalization of local training. To this scope, HarmoFL leverages special layers in the client's model architecture which perform amplitude

[4] FedFA [224] explicitly focuses on data in the form of images.

normalization and collect local amplitudes, to be then aggregated by the server to produce and employ a global one.

7.1.6 Normalization Methods

As demonstrated in several works [45, 75, 112, 191, 222], in presence of non-IID data, Batch Normalization (BN) layers [80] are detrimental for global model performance due to a mismatch among local statistics which cause shifted local features learned by clients. FedBN [112] proposes to only locally train BN's parameters (batch variance, batch mean, scale, and shift parameters) and never communicate the updates of such layers to the server for aggregation. Similarly, SiloBN [7] keeps batch variance and mean local, but communicates the scale parameter and the shift parameters for global merge. FixBN [222] proposes to regularly train, communicate and aggregate all the BN's parameters, but freezing BN's global and local statistics from a certain communication round onward. The work in [75] and the work in [45], respectively, suggest employing Group Normalization [194] or Layer Normalization [10], which do not use mini-batch statistics, in place of BN layers. FedTAN [191] tailors the local training procedure of FedAvg to ensure convergence properties in presence of BN layers at the cost of several intra-round synchronizations among client and servers.

7.1.7 Logit Calibration

A parallel line of work focuses on mitigating the detrimental effect caused by label distribution skew, a specific type of data heterogeneity, on the classification layer of neural networks. The recent work in [211] and in [154] respectively propose FedLC and DMFL that use a modified cross-entropy loss (fine-grained calibrated cross-entropy) which calibrates the local model output (i.e., logits), before feeding them to the softmax output layer, according to per-class local number of samples. A similar approach has been recently proposed in [101] with *re-weighted softmax* (WSM) to reduce catastrophic forgetting. Similarly inspired, FedRS [113] advocates the use of a *restricted softmax*, which employs rescaling factors to account for under-represented classes on the local dataset.

7.2 Server-side Optimizations

This section proposes an updated and in-depth overview of server-side approaches proposed to reduce the degradation introduced by averaging drifted model parameters/updates. Relevant lines of work include proposing different weighting schemes with respect to FedAvg, rectifying the aggregated global model with an additional phase (e.g., a distillation phase), or interpreting the model updates as "pseudo-gradient" and using an optimizer of choice – usually different from SGD – to apply them to the global model.

7.2.1 Modified Aggregation Procedure

FedNova [189] improves FedAvg in the aggregation step by considering that different clients may perform a different number of local steps (i.e., the number of mini-batches in the local training) during each round. This can result, for example, from clients holding different amounts of local examples given a fixed batch size, or from learners having approximately the same amount of local examples but different amounts of local epochs. To ensure that the global updates are not biased, FedNova designs a normalized averaging method to replace the simple FedAvg update, i.e, before updating the global model, it scales the client updates based on the local number of local steps.

Conceptually similar to the works discussed in Section 7.1.6, FedDNA [46] revisits the aggregation of statistical parameters belonging to normalization layers in the neural network (e.g., batch mean and batch variance of BN layers). In particular, FedDNA aggregates statistical parameters with an importance weighting method, and they are optimized collaboratively leveraging an adversarial learning approach, while gradient-based parameters are updated via regular FedAvg's scheme.

From another perspective, the authors of [24] suggest incorporating stochastic weight averaging (SWA) from [82] into the aggregation stage of FedAvg so to enhance the resilience of the learning process [24]. The work shows that the SWA technique is most effective when applied toward the end of the training phase, specifically after completing 75% of the total rounds.

Another body of research leverages server-side optimizers which handle aggregated model updates as "pseudo-gradient". In fact, in FedAvg, instead of uploading the brand-new model parameters, clients typically send updates to the server, which are computed as the difference between the received global model and the locally computed client model, i.e. the update rule of FedAvg can be written as:

$$w_{t+1} = \frac{1}{|S_t|}\sum_{i \in S_t} w_t^k = w_t + \frac{1}{|S|}\sum_{i \in S_t}(w_t^k - w_t) \tag{7.3}$$

with S_t being the subset of clients selected for round t. For the sake of clarity, in Eq. 7.3 the model parameters/updates are not weighted according to the number of local data samples, but it does not have an impact on the following observation: defining the client updates as $\Delta_t^k := w_t^k - w_t$ and their aggregated form as $\Delta_t := \frac{1}{|S_t|}\sum_{i \in S_t} \Delta_t^k$, we have:

$$w_{t+1} = w_t + \Delta_t = w_t - (-\Delta_t) \tag{7.4}$$

Hence, the server update rule in FedAvg is equivalent to applying SGD to the "pseudo-gradient" $-\Delta_t$ with learning rate $\eta_s = 1$. From this observation, FedAvg can be considered a specialization of a more general framework, called FedOpt, that uses SGD as a server-side optimizer [155].

Algorithm 7 presents the FedOpt framework and variants. This body of work shows that optimizers different from SGD can be used on the server, leading to the proposal of using server-side momentum or adaptive optimizers.[5] It is worth noting that, with the introduction of a server-side optimizer, η_s represents the global learning rate, which is an additional hyperparameter to tune. For server-side momentum, FedAvgM [76] uses SGD with momentum at the server side, while FedAdagrad, FedYogi, and FedAdam are the specializations of FedOpt, employing Adagrad [47, 130], Yogi [208] or Adam [95], respectively. It is worth noting that FedAvgM, FedAdam, FedYogi, and FedAdagrad do not introduce computation or communication overhead on clients with respect to FedAvg. The algorithms exhibit improved convergence rate both theoretically and empirically, also in presence of heterogeneous data. Algorithm 8 describes FedAdagrad, FedYogi and FedAdam.

Drawing connections with *out-of-distribution* generalization, Tenison *et al.* designed a Gradient Masked Aggregation (GMA) as an alternative for regular FedAvg's update aggregation [181]. In GMA, a soft mask, based on sign agreement among client updates,

[5]The client-side optimizer (ClientOpt in Algorithm 7) is supposed to be SGD.

Algorithm 7: FedOpt.

```
1 Server executes:
2     initialize $w_0$
3     for each round $t = 0, 1, 2, 3, ..$
4         $c \leftarrow max(C \times K, 1)$
5         $S_t \leftarrow$ (random set of $c$ clients)
6         for each client $k \in S_t$ in parallel
7             $\Delta_t^k \leftarrow$ ClientOpt$(k, w_t)$
8         $\Delta_t \leftarrow \sum_{i \in S_t} \frac{n_k}{n} \Delta_t^k$
9         $w_{t+1} \leftarrow$ ServerOpt$(w_t, -\Delta_t, \eta_s, t)$
```

is applied to the aggregated updates. It is worth noting that GMA can be plugged into any other algorithm as an alternative to FedAvg aggregation (e.g., into FedOpt framework).

7.2.2 Post-Aggregation Refinement

Concluding, other contributions focus on fine-tuning the global model after aggregation, before a new FL round begins. In particular, KD-based approaches have been proposed to rectify the server model by applying post-aggregation ensemble distillation. In practice, the global model is fine-tuned by mimicking the aggregated predictions of clients' models on common data. This class of solutions supposes the existence of publicly-available proxy data (semantically similar to the clients' local dataset) or leverages model generators. In detail, FedDF [115] builds the ensemble of clients' model outputs by computing their average, while FedBE [30] employs a Bayesian model ensemble to merge client predictions to improve aggregation robustness. FedAUX [159] further extends FedDF's procedure by utilizing unsupervised pre-training on auxiliary data to determine a suitable model initialization for local feature extractor. FedAUX also weighs the ensemble predictions on the proxy data based on a (ϵ, δ)-differentially private certainty score [48] of each participant model. While server-side ensemble distillation methods assume the availability of a (semantically similar) proxy dataset, FedFTG [214] conducts server-side rectification of the global model through data-free knowledge distillation, where the server adversarially trains both a generator model and the global model, and refines the latter on synthetic data. A similar data-free refinement strategy is proposed in FedZKT [213].

Algorithm 8: FedOpt: **FedAdagrad** , **FedYogi** and **FedAdam** .

```
Server executes:
    initialize $w_0$
    initialize decay parameters $\beta_1, \beta_2 \in [0, 1), v_{-1} \geq \tau^2$
    for each round $t = 0, 1, 2, 3, ..$
        $c \leftarrow max(C \times K, 1)$
        $S_t \leftarrow$ (random set of $c$ clients)
        for each client $k \in S_t$ in parallel
            $\Delta_t^k \leftarrow \text{ClientUpdate}(k, w_t)$
        $\Delta_t \leftarrow \sum_{i \in S_t} \frac{n_k}{n} \Delta_t^k$
        $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\Delta_t$
        $v_t \leftarrow v_{t-1} + \Delta_t^2$ (FedAdagrad)
        $v_t \leftarrow v_{t-1} - (1 - \beta_2)\Delta_t^2 sign(v_{t-1} - \Delta_t^2)$ (FedYogi)
        $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\Delta_t^2$ (FedAdam)
        $w_{t+1} \leftarrow w_t + \eta_s \frac{m_t}{\sqrt{v_t} + \tau}$
ClientUpdate$(k, w_t)$
    $w \leftarrow w_t$
    $\mathcal{B} \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
    for each local epoch $e$ from 1 to $E$
        for batch $b \in \mathcal{B}$
            $w \leftarrow w - \eta \nabla \ell(w; b)$
        $\Delta \leftarrow w - w_t$
    return $\Delta$ to server
```

7.3 Empirical Evaluations of SOTA Algorithms

In this section, we propose an experimental evaluation of the discussed approaches, considering both client- and server-side classes. Further contributing to the current body of knowledge, we propose a benchmark of hybrid FL approaches, emerging as a combination of client- and server-side algorithmic tweaks, discussing the tradeoffs that emerge. Our code is publicly available at https://github.com/alessiomora/fl_algorithms_non_iid_2.

7.3.1 Experimental Setup

In the following, we provide some details on the dataset used in the analysis, the partitioning strategy used to produce different levels of heterogeneity, along with the experimental parameters comprising the evaluation matrix. All the experimental simulations have been run in a Python virtual environment on a machine equipped with two NVIDIA RTX A5000 GPUs.

7.3.2 Datasets and baselines

We conducted a set of experiments on the CIFAR-100 dataset, which consists of 60.000 examples of 32x32 color images - 50.000 for training and 10.000 for testing - belonging to 100 classes. We partitioned the training set to simulate 100 clients in the federation. As standard in FL algorithm evaluations, we used the method proposed in [76] to generate the clients' shard of data. A concentration parameter α tunes the heterogeneity of local training sets. With a high α value (e.g., > 100) the distribution of labels among clients tends to be homogeneous while lowering the α value translates to local data with different amounts of examples per label until the extreme case of local examples belonging to only one label.

We performed simulations with two levels of data heterogeneity, determined by $\alpha = 0.3$ and $\alpha = 0.1$. In the first setting ($\alpha = 0.3$), we restrict the clients to have a balanced local set (i.e., the same amount of total examples) and we distribute the examples as in [94], where the authors provide the exact list of examples per client. In the latter case ($\alpha = 0.1$), we do not restrict the learners to have the same amount of total data points, and learners can have significantly unbalanced data sets. When possible, different clients do not rely on repeated examples. Fig. 7.2a and Fig. 7.2b depict the distribution of labels among the clients with varying α values.

We consider FedAvg as a baseline and benchmark a selection of state-of-the-art FL techniques among the ones presented in the first part of this paper. For client-side optimizations, we compared the following methods: FedProx, FedNTD, FedGKD, FedMLB, FedDyn, FedSAM, FedFA. For server-side optimizations, we considered FedAvgM and FedAdam. Also, we coupled promising compatible approaches, i.e. FedAdam + FedMLB, FedAvgM + FedMLB, FedDyn + FedMLB, FedAvgM + FedSAM, FedAdam + FedSAM, FedAvgM + FedFA, FedAdam + FedFA, to assess the gain deriving from both client- and server-side optimizations. While preserving the general nature of our study and without loss of generality, we have excluded from the analysis works that either assume the existence of (semantically-similar) additional public datasets or approaches known to be more bandwidth-hungry in the upload link when compared to FedAvg or techniques that do not disclose their code base and/or did not include the hyperparameter tuning procedure, or that do not use standard classification tasks in their evaluation.

7.3.3 Implementation details

Similar to [2, 94], in all the experiments, SGD with a learning rate fixed to 0.1 is used as local optimizer, and the global learning rate is set to 1.0, except for FedAdam, which used 0.01 for both local and global learning rate, and for FedAdam + FedSAM, which used 0.1 for local and 0.01 for global learning rate. Local epochs are fixed to 5 and a random fraction of 0.05 (5 %) clients are selected per round. A weight decay with a factor of 0.001 is applied to limit local overfitting. Local epochs are fixed to 5. Local batch size is determined so that each client performs 50 local updates. Gradient clipping is performed to stabilize local training. The local learning rate is exponentially decayed with a factor of 0.998 similar to the work in [2, 94]. The model architecture used in our experiments is ResNet-18 [66], but we have replaced the batch normalization layer with group normalization as suggested in [75, 94]. We used random rotation, horizontal flip, and random crop as preprocessing layers. For a fair comparison, seeds are used to select a set of random clients at each round, to perform local data preprocessing, and to initialize client models; different algorithms are compared using the same set of seeds on more runs. For ease of readability, we report a distilled set of parameters in Tab.7.3.

For algorithmic-specific hyperparameters, we proceeded as follows:

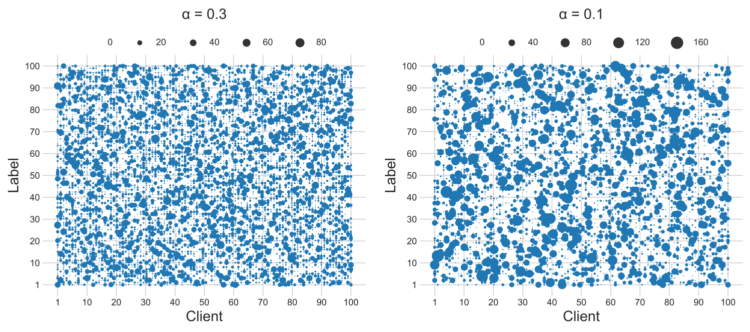
- For FedProx we tuned μ in $\{0.1, 0.01, 0.001\}$. μ controls the weight of the proximal term in the local objective function. We selected $\mu = 0.01$ for $\alpha = 0.3$ and $\mu = 0.001$ for $\alpha = 0.1$.
- For FedGKD we set γ to 0.2, as in the original paper. γ controls the weight of the KD-based term in the local objective function.
- For FedNTD we selected λ in $\{0.3, 1.0\}$.

- For FedDyn we set α_{dyn} equal to 0.1 as in the original paper.
- For FedMLB λ_1 and λ_2 are both set to 1 (λ_1, λ_2 weight the impact of the hybrid cross-entropy loss and the KD-based loss). 5 blocks are considered, formed as in the original paper, where conv1, conv2_x, conv3_x, conv4_x, conv5_x and the fully connected layer constitutes a single block.
- For FedSAM we tuned ρ in {0.5, 0.1, 0.05, 0.01} and we selected 0.05 as best value. ρ defines the neighborhood size for seeking flat minima.
- For FedFA we augment the clients' ResNet18 architecture with five FA layers, one after each ResNet block. The server aggregates the FedFA's channel-wise feature statistics and uses a regular ResNet18 architecture for test. FA layers are tuned as in the original paper [224].
- For FedAvgM we tuned the momentum parameter among $\{0.4, 0.6, 0.8, 0.9\}$, and selected 0.6 for $\alpha = 0.1$ and 0.8 for $\alpha = 0.3$.
- For FedAdam we set τ (a constant for numerical stability) equal to 0.001 as in [94, 155].

Table 7.3 reports the per-algorithm hyperparameter selection.

Table 7.3: Hyperparameter tuning of evaluated algorithms.

Method	Selected ($\alpha = 0.3$, bal.)	Selected ($\alpha = 0.1$, unbal.)
FedProx	μ = 0.01	μ = 0.001
FedNTD	λ = 0.3	
FedGKD (M=1)	γ = 0.2	
FedGKD (M=5)	γ = 0.2	
FedDyn	α_{dyn} = 0.1	
FedMLB	λ_1 = 1, λ_2 = 1	
FedSAM	ρ = 0.05	
FedAdam	τ = 0.001, β_1=0.9, β_2=0.999	
FedAvgM	momentum = 0.8	momentum = 0.6



(a) $\alpha = 0.3$, balanced datasets. (b) $\alpha = 0.1$, unbalanced datasets.

Figure 7.2: Label distribution with different levels of heterogeneity. (a) Distribution of labels (0-99) on clients (100 clients) with concentration parameter set to 0.3. Local datasets have the same cardinality. This is the exact same setting of [94]. (b) Distribution of labels (0-99) on clients (100 clients) with concentration parameter set to 0.1.

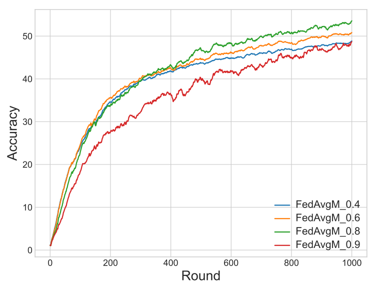


Figure 7.3: FedAvgM accuracy with different momentum parameters, $\alpha = 0.3$.

The algorithms are implemented in Python using the TensorFlow library. It is worth noting that most of the code available for such algorithms is implemented leveraging the Pytorch library, hence the accompanying code of this paper constitutes a valuable contribution to the FL community. The code base also provides the implementation of MOON, but the latter is not included in the comparison analysis since it locally modifies the model architecture by injecting a 2-layer head.

Table 7.4, Tab.7.5 and Tab. 7.6 respectively report the results of the client-side, server-side and hybrid FL methods, for different levels of data heterogeneity. The arrows denote whether the higher (accuracy) or the lower (rounds) is better. Since both final accuracy and convergence speed are relevant indicators for federated learning [168], we report the performance attained at different rounds and the number of needed communication

Table 7.4: Comparison among considered client-side methods on CIFAR-100 within different federated learning settings. The accuracy at the target round and the number of communication rounds to reach the target test accuracy are based on smoothed average with parameter 0.9.

Method	$\alpha = 0.3$ (balanced)					$\alpha = 0.1$ (unbalanced)			
	Accuracy (%, ↑)		Rounds (↓)			Accuracy (%, ↑)		Rounds (↓)	
	500R	1000R	25%	45%	47%	500R	1000R	25%	39%
FedAvg	42.78	47.02	127	756	965	33.06	39.83	309	889
FedProx	42.36	46.91	128	754	1000+	33.21	39.83	309	889
FedNTD	42.38	46.66	123	764	1000+	33.91	40.67	307	807
FedGKD (M=1)	43.13	47.64	122	657	935	33.74	40.56	309	806
FedGKD (M=5)	42.61	46.48	123	663	1000+	34.07	40.55	307	802
FedDyn	48.90	58.11	89	334	414	36.39	48.02	269	536
FedMLB	49.21	55.60	122	390	440	31.97	42.68	368	743
FedSAM	43.62	48.54	143	575	729	34.12	42.83	321	708
FedFA	46.54	52.08	149	443	548	37.48	47.17	310	549

Table 7.5: Comparison among considered server-side methods on CIFAR-100 in two different federated learning settings. The accuracy of the target round and the number of communication rounds to reach the target test accuracy are based on smoothed average with parameter 0.9.

Method	$\alpha = 0.3$ (balanced)					$\alpha = 0.1$ (unbalanced)			
	Accuracy (%, ↑)		Rounds (↓)			Accuracy (%, ↑)		Rounds (↓)	
	500R	1000R	25%	45%	55%	500R	1000R	25%	39%
FedAvg	42.78	47.02	127	756	1000+	33.06	39.83	309	889
FedAvgM	42.99	53.55	113	464	1000+	35.38	42.51	250	724
FedAdam	44.56	53.09	139	422	1000+	41.76	47.42	193	430

rounds to reach a target accuracy. For the solutions that do not achieve the target accuracy in 1000 rounds, we use the notation 1000+.

Table 7.7 reports the local execution time of the best performing client-side approaches (i.e., FedDyn, FedMLB, FedSAM, FedFA) with respect to FedAvg. The reported time (in seconds) is the average local training time across 5 clients per round, over 150 rounds, considering the setting ($\alpha = 0.3$, balanced), 500 local examples, 5 local epochs, and 50

Table 7.6: Comparison among hybrid methods on CIFAR-100 in two different federated learning settings. The accuracy at the target round and the number of communication rounds to reach the target test accuracy are based on smoothed average with parameter 0.9.

Method	$\alpha = 0.3$ (balanced)					$\alpha = 0.1$ (unbalanced)			
	Accuracy (%, ↑)		Rounds (↓)			Accuracy (%, ↑)		Rounds (↓)	
	500R	1000R	25%	45%	55%	500R	1000R	25%	39%
FedAvg	42.78	47.02	127	756	1000+	33.06	39.83	309	889
FedAvgM + FedFA	48.56	55.25	113	329	849	**41.94**	49.71	**221**	**413**
FedAvgM + FedSAM	49.43	56.75	122	330	751	39.01	47.27	235	504
FedAdam + FedSAM	49.58	56.55	144	265	790	38.88	49.65	279	501
FedAvgM + FedMLB	52.56	58.55	133	333	636	35.30	50.21	329	603
FedAdam + FedMLB	51.22	58.17	124	357	742	39.6	50.74	242	476
FedDyn + FedMLB	**58.97**	**63.31**	**81**	**222**	**264**	34.92	**54.59**	341	573

Table 7.7: Execution time (in seconds) of best performing client-side algorithms and overhead with respect to FedAvg, setting ($\alpha = 0.3$, balanced).

Method	Time (s, ↓)	wrt FedAvg (↓)
FedAvg	10.91	
FedDyn	11.07	$\approx$
FedFA	15.91	1.45x
FedSAM	17.64	1.61x
FedMLB	25.25	2.31x

local updates. The per-round pseudo-random selection of clients is ruled by the same set of seeds for all the measurements to ensure fairness. Clients run sequentially in our centralized simulations. We use the results in Tab. 7.7 as a proxy for qualitatively assessing the computation overhead, with respect to FedAvg, introduced by the considered state-of-the-art algorithms.

7.3.4 Client-side Methods

Fig. 7.4a and Fig. 7.4f show the test accuracy for the considered client-side methods. As it is shown, FedProx has a similar performance to FedAvg, while being very sensitive to its

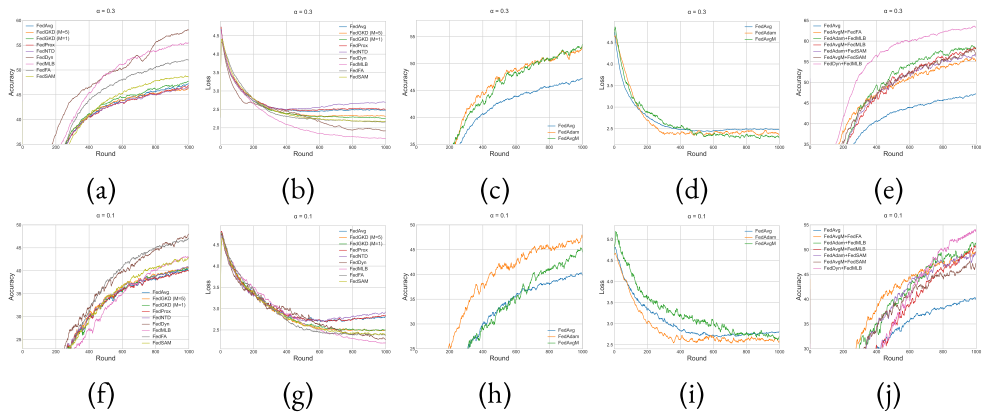


Figure 7.4: The figure depicts the global model's performance (accuracy and loss), also reported in Tab. 7.4, Tab. 7.5 and Tab. 7.6, on test data across 1000 communication rounds. The first row of charts refers to the setting ($\alpha = 0.3$, balanced datasets), the second one to ($\alpha = 0.1$, unbalanced datasets). (a, f) Accuracy of considered client-side methods with different levels of data heterogeneity. (b, g) Loss of considered client-side methods with different level of data heterogeneity. (c, h) Accuracy of considered server-side methods with different level of data heterogeneity. (d, i) Loss of considered server-side methods with different level of data heterogeneity. (e, j) Accuracy of considered combined methods with different level of data heterogeneity.

additional hyperparameter (reported best results with μ = 0.01 for α = 0.3 and with μ = 0.001 for α = 0.3) since the proximal term directly scales with a measure of model weight divergence. Furthermore, it is worth noting that FedProx has been also designed to handle a different number of local computations (e.g., different amounts of local epochs), which is not the setting we reproduced (all the clients perform the same number of local updates in our experiments).

One interesting result is that the baseline methods that use the global model output in a KD-based framework (FedGKD, FedNTD) only show negligible gains or even degrade global the model accuracy. As also claimed in [68], this can be explained by the fact that the predictions of the global model can be misleading during training, especially in the first communication rounds, due to inaccurate predictions. Hence, mimicking the response of the global model is suboptimal and can hamper local training. Contrarily, FedMLB is more robust in presence of heterogeneous data and largely outperforms the other client-side KD-based methods, even though it results to be more effective on the setting ($\alpha = 0.3$, balanced) with a relative increase of 18% with respect to FedAvg which drops to slightly more than 7% for ($\alpha = 0.1$, balanced). This result confirms that local-global distillation is motivated: the guidance of the global model can be significantly beneficial to tackle data heterogeneity. At the same time, in response-based local-global distillation (FedGKD, FedNTD), locally learned intermediate features may drift apart, and approaches that also pose constraints to intermediate representations are more effective. From a different perspective, the global model should be better at producing general intermediate features, and its influence on local training is beneficial.

However, FedMLB introduces significant local computation and memory overhead, as well as more local execution time (see Tab. 7.7), which obviously translates to an increase in energy consumption. While not representing an issue in cross-silo federations, such overhead may preclude the participation of resource-constrained nodes in cross-device settings.

It is worth mentioning that FedGKD (with M = 5 historical models) shows a slight improvement over FedAvg in the $\alpha = 0.1$ setting with limited memory and computation overhead. However, it requires 2 times the server-client communication costs compared to FedAvg and the other methods, since the server also transmits the aggregated historical model. Fig. 7.4b and Fig. 7.4g depict the global model loss on test data for client-side approaches, and the trends that emerge are quite consistent with the analysis conducted on global model accuracy. An interesting aspect is that FedGKD reaches lower loss values with respect to FedAvg, which however translates either to a slight improvement or even in degradation in accuracy. Similarly, FedMLB results in lower test loss with respect to FedDyn, which conversely achieves the best accuracy, in both the heterogeneity levels.

FedFA exhibits significant improvements in generalization over FedAvg, at negligible cost of memory overhead and limited overhead of local execution time – FedFA resulted to be the solution with the lowest execution time after FedDyn (see Table 7.7). It is worth highlighting that FedFA requires to modify the local model architecture (FedFA leverages special layers that inject feature noise) and requires disclosing additional local information about aggregated statistics of data – which have negligible communication cost.

FedSAM results to be more effective in presence of higher heterogeneity levels ($\alpha = 0.1$, unbalanced), surpassing FedMLB in global model accuracy both at 500R and 1000R. Contextually, FedSAM requires longer local training time with respect to FedDyn and FedFA which however exhibit better generalization, while not requiring prior knowledge about the federation or modifications to the local model architecture.

With respect to client-side approaches, FedDyn shows impressive results in both the considered settings and achieves the best accuracy after 1000 learning rounds (respectively 58.11% and 48.02%), while introducing marginal overhead in local execution time (see Table 7.7). Recall that FedDyn is classified among client-side approaches, even though it maintains a server-side state which is vital to the algorithm. Its memory requirements, however, are at least doubled with respect to FedAvg to be attributed to the computation of its penalty term; furthermore, FedDyn supposes a prior knowledge about the number of total clients, which may be not a realistic assumption in certain circumstances.

7.3.5 Server-side Methods

When considering the server-side approaches - FedAvgM and FedAdam - they exhibit significant improvements over FedAvg in both the considered settings (see Tab. 7.5, Fig. 7.4c, Fig. 7.4h). The global model loss on test data confirms the trends of global model accuracy (see Fig. 7.4d, 7.4i). Overall, FedAdam achieves the best results, performing slightly worse than FedAvgM in ($\alpha = 0.3$, balanced) but showing a significant improvement in the more extreme setting. Also, FedAdam is consistently faster in achieving target accuracy levels. We note once again that, differently from client-side methods, FedAvgM and FedAdam do not introduce client-side overhead. Hwever, they require additional hyperparameters to be tuned, i.e. the server-side learning rate and other hyperparameters specific to the optimizer (e.g., momentum for FedAvgM). Although their best configurations outperform FedAvg, they seem quite sensitive to such hyperparameters, as also shown in [155]. For the sake of clarity, in Fig. 7.3, we report the performance of FedAvgM with different momentum parameters.

7.3.6 Hybrid Methods

Table 7.6 reports the results of hybrid methods, derived as a combination of the most promising client- and server-ide approacheswhich are: FedAvgM + FedMLB, FedAdam + FedMLB, FedDyn + FedMLB, FedAvgM + FedSAM, FedAdam + FedSAM, and FedAvgM + FedFA. Fig. 7.4e and Fig. 7.4j depict the achieved accuracy of the considered combined solutions.

The first two methods exhibit very similar performance, achieving approximate levels of accuracy, while the Adam-based one converges faster. Overall, the combination Fed-

Dyn + FedMLB attains the best generalization performances. For the setting ($\alpha = 0.3$, balanced), FedDyn + FedMLB results to be the best method in convergence speed and in maximum accuracy achieving 55% accuracy in 264 communication rounds, with FedAvg never achieving a comparable accuracy even in 1000 communication rounds. For ($\alpha = 0.1$, unbalanced), FedDyn + FedMLB converges slower than other solutions but shows the best final accuracy. It is worth noting that, while outperforming all the other considered solutions, FedDyn + FedMLB requires the combined overhead of the two solutions, resulting in a non-negligible increase in memory usage and local execution time for client devices (see Tab. 7.7), hence at the expense of greater energy consumption, as well as supposing FedDyn's prior knowledge about the federation. FedAvgM + FedFA shows an impressive slope in the first part of the training, resulting as the faster method to achieve the target levels of accuracy (25%, 39%) in the most extreme setting. FedSAM seems to be most effective when coupled with server-side optimization, bridging the gap with other client-side methods.

7.3.7 Discussion

The analysis shows that improving FedAvg's robustness to data heterogeneity is a challenging task. Superior approaches, e.g. FedDyn or FedMLB, usually introduce additional overhead (memory, execution time and energy consumption, communication costs), assume prior domain knowledge or leverage proxy datasets. Overall, the results confirm that (1) approaches rooted in theory, like FedDyn, are extremely effective, (2) transferring knowledge between global and local models, in an effort to limit catastrophic forgetting, effectively improves generalization, with utilizing intermediate features being crucial, (3) seeking for flatter local minima also translates to improved robustness of the global model confirming that SAM-trained local models are more amenable to averaged aggregation, and (4) feature-augmentation layers can be an additional architectural element to be considered when designing neural networks for use in federated environments. Finally, it is evident the benefit of combining client- and server-side techniques so to boost the effectiveness of single-sided approaches.

7.4 Concluding Remarks

In this Chapter, we have reviewed state-of-the-art algorithms to tackle the degradation in global model performance introduced by data heterogeneity. We classified the methods in two macro categories, client- and server-side optimizations, to then clustering them in finer subgroups driven by the same inspirations or observations. In the second part of this Chapter, we have provided experimental evaluations of a subset of the reviewed solutions in challenging simulated levels of data heterogeneity. In line with the most recent literature in the field, we showed that some solutions are more effective than others, but these benefits may come at an unsustainable cost for client devices. The code we used for experiment is available at `https://github.com/alessiomora/fl_algorithms_non_iid_2`.

8 Future Directions and Conclusions

8.1 Future Directions

To conclude, in the next subsections, we will present some open challenges that will likely influence the incoming future of decentralized learning systems, by also sketching possible and most promising directions for future research.

8.1.1 Towards Model Heterogeneity

As we have seen in Chapter 6, the literature comprises a plethora of strategies to enable model heterogeneity; however, they rely on unpractical assumptions or show significantly reduced effectiveness with respect to methods that communicate model parameters. We expect work to narrow such a performance gap with traditional algorithms. This degree of freedom would further favor the collaboration among institutions — under the perspective of intellectual property related to the tailored model architecture — and can be also leveraged to favor the inclusion of more resource-constrained edge devices in the learning process. In addition, methods that enable model heterogeneity may be more amenable to model personalization, representing an interesting perspective to be deepened.

8.1.2 Towards Model Personalization

The literature review, the discussions and the empirical evaluations present in this Thesis focus on methods that aim at building a global model that generalizes well on unseen data, either reducing the communication cost to train it, including resource-constrained devices in the process, guaranteeing certain privacy levels, or being robust to data heterogeneity. However, recently, there has been an increasing focus in the field on personalizing the global model to better fit the data of individual participants, and it is emerging as a significant factor in evaluating algorithms or for designing new ones [86, 144]. Particularly relevant for this Thesis, it would be to extend the treatment in Chapter 7 by evaluating the state-of-the-art methods to tackle data heterogeneity under personalization metrics (e.g., after a certain number of communication rounds, how effective the global model is in fitting local data, using a simulated local test data which reflects the local distribution). In particular, the focus would be to understand whether KD can be used to boost model personalization as it has been done to increase robustness of the global model to data heterogeneity.

8.1.3 Evaluating Federated Algorithms on Real Testbeds

Almost all the approaches proposed in literature and referenced in this Thesis evaluate their novel solutions on simulated federation of clients, running them in a centralized manner and leveraging datacenter-like hardware, such as powerful GPUs. However, often such evaluations do not take into account the actual possible overhead on client devices, which is a very relevant aspect for cross-device settings, as we pointed out several times in this document. Usually, the performance analysis focuses on metrics such as the global model generalization (or personalizaiton) ability, the loss on test data either from a local or global distribution, comparing them with respect to well-known baselines for a baseline algorithm, e.g. FedAvg for synchronous, star-shaped FL. We claim that evaluations of state-of-the-art solutions should include empirical evaluations of possible overhead on clients, such as execution time, computational cost and energy consumption, measured on hardware representative of real edge devices. Most importantly for this Thesis, the empirical results in Chapter 7 could be extended to include these kind of metrics as future work, which would be novel for the field as well as necessary for the adoption. From another perspective, the datasets usually employed in experiments are federated variants of well-known regular datasets, such as EMNIST, CIFAR10, and CIFAR100. Such simulated client distributions, while being very practical for benchmarking new algorithms, may be not fully representative of actual federated environments, calling for new generations of datasets, collected in real-life exemplary use case.

8.1.4 Rethinking the Traditional ML Workflow for Federated Learning

The literature explored in this survey proposes solutions to the main challenges of employing federated learning systems in real-world scenarios. However, most works suppose that the hyperparameters (e.g., the neural network's architecture, regularization techniques, and optimizers) of the model to be trained have been already established, and typically the focus is not about the tuning of their determination. Furthermore, decentralized learning systems introduce additional algorithm-specific hyperparameters (e.g., the number of local epochs or the number of participants involved per round) that significantly influence the performance of the adopted solution. While in cloud-centric DL it is feasible to run many rounds of training to empirically search the hyperparameters space towards optimality, this strategy is probably infeasible for cross-silo settings and surely incompatible with cross-device settings. Hence, we expect that hyperparameter optimization that targets the communication and computation overhead on the devices that compose the federation, and not only aiming at the best accuracy of models as happens in datacenter optimizations, will gain traction, by fostering the development of easy-to-tune and/or auto-tuning algorithms for federated settings.

Another relevant phase of the traditional workflow in cloud-centric ML, which is reshaped by the design of decentralized learning systems, relates to the debugging of trained models' behaviour. In fact, preventing the access to the raw data by design does preclude modelers and practitioners from directly investigating the causes of the detected problems (e.g., investigating missclassification, noticing evident bias in the training set, identifying outliers, manually adding or adjusting labels), i.e. manual data inspection is impossible. Connected to that, the design and implementation of privacy-preserving techniques to enable the debug phase also for federated learning systems are open areas of research.

8.1.5 Going beyond Supervised Learning

It is important to underline once more that almost all the cited works in this survey suppose labeled data examples within supervised learning contexts. However, in real federated settings it could not be straightforward to automatically or to manually label data samples; while systems to favour the collection of user-annotated examples are arising (e.g., [122]), the huge amount of unlabeled raw data, that will be produced in the next years at the edge of the network, may not be adequately exploited by only supervised learning techniques. Anyway, opening up to semi-supervised [88], unsupervised or to reinforcement learning approaches would require similar issues in terms of privacy guarantees, heterogeneity, communication efficiency and scalability.

8.1.6 Towards Fully Decentralized Systems at Scale

While cross-device (star-shaped) FL is mature enough to be used in large scale applications [18] (e.g., in the realm of smartphone apps), cross-device fully decentralized solutions have not reached such mature implementations yet. As already highlighted, dealing with peer-to-peer topologies inherently adds layers of complexity with respect to the client-server paradigm; that makes it harder the implementation as well as the theoretical analysis of such systems. A very practical solution may be having a central orchestrating entity that is aware of the current topology status thanks to periodic reports provided by the federation of peers (as in [179]); in this way the orchestrator[1] can determine and dictate the (favourable) peer links to be used in exchanging model updates. In this perspective, in the short-term future research in the field, we expect growing efforts in practical (and maybe more elegant) solutions to dominate the complexity of dynamic large-scale peer-to-peer topologies, as in the case of real cross-device federated scenarios of practical usage, since fully decentralized systems bring, in principle, several advantages with respect to star-shaped solutions (e.g., no need to trust central entities, no server bottlenecks, no unique points of failure). We also note that while communication-efficient strategies can be more easily adapted from star-shaped to fully decentralized systems (e.g., [179]), this may be not so natural for non-IIDness and for privacy guarantees. Furthermore, as far as we know, poisoning has not been investigated considering such topology of participants. In short, the literature about fully decentralized learning is still in its embryonic stages: approaches to ensure formal privacy guarantees (e.g., DP-based approaches and secure aggregation adaptations) and to effectively tackle non-IIDness (e.g., [143]) have still to be thoughtfully explored and investigated before achieving the efficient implementation and deployment of an associated large-scale prototype.

[1]The orchestrator may also easily dictate the hyperparameters of the model to be trained and of the algorithm to be used.

8.2 Conclusions

In this Thesis, we firstly proposed an extensive overview of decentralized learning literature, enriched by the proposal and usage of a novel taxonomy, which has been used to analyze and classify state-of-the-art algorithmic methodologies. Then, the main issues in decentralized learning have been explored, providing a landscape on possible solutions tailored for diverse settings or architectural designs. In Chapter 4 and Chapter 5, we presented the rationale, the design and the empirical evaluation of two methods which aim at improving the communication efficiency and the robustness to system heterogeneity of FedAvg, the most spread baseline algorithm. The conceptual contribution of this Thesis also included, in Chapter 6, a novel perspective on the usage of Knowledge Distillation as a mean for alleviating issues such as data heterogeneity and system heterogeneity. Finally, in Chapter 7, we provided a vertical review on recently proposed methods to tackle the performance degradation introduced by data heterogeneity in star-shaped Federated Learning, enriched by empirical evaluations on challenging heterogeneous data distributions.

Acronyms

CCPA	California Consumer Privacy Act
CNN	Convolutional Neural Network
DL	Deep Learning
DP	Differential Privacy
FL	Federated Learning
GDPR	European General Data Protection Regulation
GL	Gossip Learning
HIPAA	Health Insurance Portability and Accountability Act
IID	Identically and Independently Distributed
IIoT	Industrial Internet of Things
IoT	Internet of Things
KD	Knowledge Distillation
MEC	Multi-access Edge Computing
ML	Machine Learning
MPC	Multi-party Computation
NN	Neural Network
SGD	Stochastic Gradient Descent
SSTC	Structured Sparse Ternary Compression
STC	Sparse Ternary Compression
VPN	Virtual Private Network

Bibliography

1. M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 308–318.
2. D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, and V. Saligrama. "Federated learning based on dynamic regularization". *arXiv preprint arXiv:2111.04263*, 2021.
3. A. Afonin and S. P. Karimireddy. "Towards Model Agnostic Federated Learning Using Knowledge Distillation". *arXiv preprint arXiv:2110.15210*, 2021.
4. N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan. "cpSGD: Communication-efficient and differentially-private distributed SGD". In: *Advances in Neural Information Processing Systems*. 2018, pp. 7564–7575.
5. A. F. Aji and K. Heafield. "Sparse communication for distributed gradient descent". *arXiv preprint arXiv:1704.05021*, 2017.
6. D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. "QSGD: Communication-efficient SGD via gradient quantization and encoding". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1709–1720.
7. M. Andreux, J. O. du Terrail, C. Beguier, and E. W. Tramel. "Siloed Federated Learning for Multi-centric Histopathology Datasets". In: *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*. Springer. 2020, pp. 129–139.
8. R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton. "Large scale distributed neural network training through online distillation". *arXiv preprint arXiv:1804.03235*, 2018.

9. P. Auer, N. Cesa-Bianchi, and P. Fischer. "Finite-time analysis of the multiarmed bandit problem". *Machine learning* 47:2-3, 2002, pp. 235–256.
10. J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer normalization". *arXiv preprint arXiv:1607.06450*, 2016.
11. E. Bagdasaryan, O. Poursaeed, and V. Shmatikov. "Differential privacy has disparate impact on model accuracy". In: *Advances in Neural Information Processing Systems*. 2019, pp. 15453–15462.
12. E. Bakopoulou, B. Tillman, and A. Markopoulou. "A Federated Learning Approach for Mobile Packet Classification". *arXiv preprint arXiv:1907.13113*, 2019.
13. P. Bellavista, L. Foschini, and A. Mora. "Communication-Efficient Heterogeneous Federated Dropout in Cross-device Settings". In: *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2021, pp. 1–6.
14. P. Bellavista, L. Foschini, and A. Mora. "Decentralised Learning in Federated Deployment Environments: A System-Level Survey". *ACM Computing Surveys (CSUR)* 54:1, 2021, pp. 1–38.
15. J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. "signSGD: Compressed optimisation for non-convex problems". *arXiv preprint arXiv:1802.04434*, 2018.
16. M. Blot, D. Picard, M. Cord, and N. Thome. "Gossip training for deep learning". *arXiv preprint arXiv:1611.09726*, 2016.
17. M. Blum and S. Micali. "How to generate cryptographically strong sequences of pseudorandom bits". *SIAM journal on Computing* 13:4, 1984, pp. 850–864.
18. K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, et al. "Towards federated learning at scale: System design". *arXiv preprint arXiv:1902.01046*, 2019.
19. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. "Practical secure aggregation for privacy-preserving machine learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1175–1191.

20. K. Bonawitz, F. Salehi, J. Konečnỳ, B. McMahan, and M. Gruteser. "Federated learning with autotuned communication-efficient secure aggregation". *arXiv preprint arXiv:1912.00131*, 2019.

21. N. Bouacida, J. Hou, H. Zang, and X. Liu. "Adaptive Federated Dropout: Improving Communication Efficiency and Generalization for Federated Learning". *arXiv preprint arXiv:2011.04050*, 2020.

22. T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi. "Federated learning of predictive models from federated electronic health records". *International journal of medical informatics* 112, 2018, pp. 59–67.

23. C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.

24. D. Caldarola, B. Caputo, and M. Ciccone. "Improving Generalization in Federated Learning by Seeking Flat Minima". In: *Proc. of European Computer Vision Conference*. Springer. 2022, pp. 654–672.

25. S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. "Leaf: A benchmark for federated settings". *arXiv preprint arXiv:1812.01097*, 2018.

26. S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar. "Expanding the reach of federated learning by reducing client resource requirements". *arXiv preprint arXiv:1812.07210*, 2018.

27. S. of California Department of Justice. *California Consumer Privacy Act (CCPA)*. URL `https://oag.ca.gov/privacy/ccpa`. Accessed on May 2020.

28. Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng. "TiFL: A Tier-based Federated Learning System". *arXiv preprint arXiv:2001.09249*, 2020.

29. H. Chang, V. Shejwalkar, R. Shokri, and A. Houmansadr. "Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer". *arXiv preprint arXiv:1912.11279*, 2019.

30. H.-Y. Chen and W.-L. Chao. "Fedbe: Making bayesian model ensemble applicable to federated learning". *arXiv preprint arXiv:2009.01974*, 2020.

31. M. Chen, R. Mathews, T. Ouyang, and F. Beaufays. "Federated Learning Of Out-Of-Vocabulary Words". *arXiv preprint arXiv:1903.10635*, 2019.

32. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

33. W. Chen, K. Bhardwaj, and R. Marculescu. "Fedmax: Mitigating Activation Divergence for Accurate and Communication-efficient Federated Learning". In: *Proc. of Machine Learning and Knowledge Discovery in Databases: European Conference*. Springer. 2021, pp. 348–363.

34. X. Chen, T. Chen, H. Sun, Z. S. Wu, and M. Hong. "Distributed Training with Heterogeneous Data: Bridging Median and Mean Based Algorithms". *arXiv preprint arXiv:1906.01736*, 2019.

35. Y. Chen, X. Sun, and Y. Jin. "Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation". *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

36. S. Cheng, J. Wu, Y. Xiao, and Y. Liu. "Fedgems: Federated learning of larger server models via selective knowledge fusion". *arXiv preprint arXiv:2110.11027*, 2021.

37. Cisco. *Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper*. URL `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html`. Accessed on April 2020.

38. G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. "EMNIST: Extending MNIST to handwritten letters". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.

39. J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al. "Large scale distributed deep networks". In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

40. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". *arXiv preprint arXiv:1810.04805*, 2018.

41. I. Diakonikolas, G. Kamath, D. M. Kane, J. Li, A. Moitra, and A. Stewart. "Being robust (in high dimensions) can be practical". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 999–1008.

42. W. Diffie and M. Hellman. "New directions in cryptography". *IEEE transactions on Information Theory* 22:6, 1976, pp. 644–654.

43. T. V. Doan, Z. Fan, G. T. Nguyen, H. Salah, D. You, and F. H. Fitzek. "Follow Me, If You Can: A Framework for Seamless Migration in Mobile Edge Cloud". *IEEE INFOCOM Workshops*, 2020, pp. 1178–11183.

44. A. Dosovitskiy and T. Brox. "Inverting visual representations with convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4829–4837.

45. Z. Du et al. "Rethinking Normalization Methods in Federated Learning". In: *Proc. of the 3rd International Workshop on Distributed Machine Learning*. 2022, pp. 16–22.

46. J.-H. Duan, W. Li, and S. Lu. "FedDNA: Federated Learning with Decoupled Normalization-layer Aggregation for Non-iid Data". In: *Proc. of Machine Learning and Knowledge Discovery in Databases*. Springer. 2021, pp. 722–737.

47. J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* 12:7, 2011.

48. C. Dwork, A. Roth, et al. "The algorithmic foundations of differential privacy". *Foundations and Trends® in Theoretical Computer Science* 9:3–4, 2014, pp. 211–407.

49. EU. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. URL `https://eur-lex.europa.eu/legal-content/EN/TXT/`.

50. R. Fantacci and B. Picano. "Federated learning framework for mobile edge computing networks". *CAAI Transactions on Intelligence Technology* 5:1, 2020, pp. 15–21.

51. P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. "Sharpness-aware Minimization for Efficiently Improving Generalization". In: *Proc. of International Conference on Learning Representations*. 2021.

52. Y. Fu, H. Wang, K. Xu, H. Mi, and Y. Wang. "Mixup Based Privacy Preserving Mixed Collaboration Learning". In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2019, pp. 275–2755.

53. C. Fung, C. J. Yoon, and I. Beschastnikh. "Mitigating sybils in federated learning poisoning". *arXiv preprint arXiv:1808.04866*, 2018.

54. R. C. Geyer, T. Klein, and M. Nabi. "Differentially private federated learning: A client level perspective". *arXiv preprint arXiv:1712.07557*, 2017.

55. L. Giaretta and Š. Girdzijauskas. "Gossip learning: Off the beaten path". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 1117–1124.

56. X. Gong, A. Sharma, S. Karanam, Z. Wu, T. Chen, D. Doermann, and A. Innanje. "Ensemble Attention Distillation for Privacy-Preserving Federated Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 15076–15086.

57. I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". *arXiv preprint arXiv:1312.6211*, 2013.

58. J. Gou, B. Yu, S. J. Maybank, and D. Tao. "Knowledge distillation: A survey". *International Journal of Computer Vision* 129:6, 2021, pp. 1789–1819.

59. O. Gupta and R. Raskar. "Distributed learning of deep neural network over multiple agents". *Journal of Network and Computer Applications* 116, 2018, pp. 1–8.

60. M. Hao, H. Li, G. Xu, S. Liu, and H. Yang. "Towards Efficient and Privacy-Preserving Federated Deep Learning". In: *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE. 2019, pp. 1–6.

61. A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. "Federated learning for mobile keyboard prediction". *arXiv preprint arXiv:1811.03604*, 2018.

62. C. Hardy, E. Le Merrer, and B. Sericola. "Gossiping GANs". In: 2018.

63. V. Hartmann and R. West. "Privacy-Preserving Distributed Learning with Secret Gradient Descent". *arXiv preprint arXiv:1906.11993*, 2019.

64. C. He, M. Annavaram, and S. Avestimehr. "Group knowledge transfer: Federated learning of large cnns at the edge". *Advances in Neural Information Processing Systems* 33, 2020, pp. 14068–14080.

65. C. He, C. Tan, H. Tang, S. Qiu, and J. Liu. "Central server free federated learning over single-sided trust social networks". *arXiv preprint arXiv:1910.04956*, 2019.

66. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

67. Y. He, Y. Chen, X. Yang, H. Yu, Y.-H. Huang, and Y. Gu. "Learning Critically: Selective Self-Distillation in Federated Learning on Non-IID Data". *IEEE Transactions on Big Data*, 2022.

68. Y. He, Y. Chen, X. Yang, Y. Zhang, and B. Zeng. "Class-Wise Adaptive Self Distillation for Heterogeneous Federated Learning", 2022.

69. U. D. of Health & Human Services. *The HIPAA Privacy Rule*. URL `https://www.hhs.gov/hipaa/for-professionals/privacy/index.html`. Accessed on May 2020.

70. I. Hegedűs, G. Danner, and M. Jelasity. "Gossip Learning as a Decentralized Alternative to Federated Learning". In: *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2019, pp. 74–90.

71. G. Hinton, O. Vinyals, and J. Dean. "Distilling the knowledge in a neural network". *arXiv preprint arXiv:1503.02531*, 2015.

72. B. Hitaj, G. Ateniese, and F. Perez-Cruz. "Deep models under the GAN: information leakage from collaborative deep learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 603–618.

73. S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. I. Venieris, and N. D. Lane. "FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout". *arXiv preprint arXiv:2102.13451*, 2021.

74. W. Hou, D. Wang, and X. Chen. "Generate Images with Obfuscated Attributes for Private Image Classification". In: *International Conference on Multimedia Modeling*. Springer. 2020, pp. 125–135.

75. K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons. "The non-iid data quagmire of decentralized machine learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4387–4398.

76. T.-M. H. Hsu, H. Qi, and M. Brown. "Measuring the effects of non-identical data distribution for federated visual classification". *arXiv preprint arXiv:1909.06335*, 2019.

77. L. Hu, H. Yan, L. Li, Z. Pan, X. Liu, and Z. Zhang. "MHAT: an efficient model-heterogenous aggregation training scheme for federated learning". *Information Sciences* 560, 2021, pp. 493–503.

78. G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. "Deep networks with stochastic depth". In: *European conference on computer vision*. Springer. 2016, pp. 646–661.

79. L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu. "LoAdaBoost: Loss-Based AdaBoost Federated Machine Learning on medical Data". *arXiv preprint arXiv:1811.12629*, 2018.

80. S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proc. of the International Conference on Machine Learning*. pmlr. 2015, pp. 448–456.

81. S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto. "Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data". *arXiv preprint arXiv:2008.06180*, 2020.

82. P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. "Averaging weights leads to wider optima and better generalization". *arXiv preprint arXiv:1803.05407*, 2018.

83. E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim. “Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data”. *arXiv preprint arXiv:1811.11479*, 2018.

84. J. Jiang, L. Hu, C. Hu, J. Liu, and Z. Wang. “BACombo—Bandwidth-Aware Decentralized Federated Learning”. *Electronics* 9:3, 2020, p. 440.

85. M. Jiang, Z. Wang, and Q. Dou. “Harmofl: Harmonizing Local and Global Drifts in Federated Learning on Heterogeneous Medical Images”. In: *Proc. of AAAI Conference on Artificial Intelligence*. Vol. 36. 1. 2022, pp. 1087–1095.

86. H. Jin, D. Bai, D. Yao, Y. Dai, L. Gu, C. Yu, and L. Sun. “Personalized Edge Intelligence via Federated Self-Knowledge Distillation”. *IEEE Transactions on Parallel and Distributed Systems* 34:2, 2022, pp. 567–580.

87. R. Jin, Y. Huang, X. He, H. Dai, and T. Wu. “Stochastic-Sign SGD for Federated Learning with Theoretical Guarantees”. *arXiv preprint arXiv:2002.10940*, 2020.

88. Y. Jin, X. Wei, Y. Liu, and Q. Yang. “A Survey towards Federated Semi-supervised Learning”. *arXiv preprint arXiv:2002.11545*, 2020.

89. P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. “Advances and open problems in federated learning”. *arXiv preprint arXiv:1912.04977*, 2019.

90. M. Kamp, L. Adilova, J. Sicking, F. Hüger, P. Schlicht, T. Wirtz, and S. Wrobel. “Efficient decentralized deep learning by dynamic model averaging”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, pp. 393–409.

91. S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh. “SCAFFOLD: Stochastic controlled averaging for on-device federated learning”. *arXiv preprint arXiv:1910.06378*, 2019.

92. K. Karras, E. Pallis, G. Mastorakis, Y. Nikoloudakis, J. Mongay Batalla, C. X. Mavromoustakis, and E. K. Markakis. “A Hardware Acceleration Platform for AI-Based Inference at the Edge”. *Circuits Syst. Signal Process.* 39:2, 2020, pp. 1059–1070.

93. H. Kim, J. Park, M. Bennis, and S.-L. Kim. “On-device federated learning via blockchain and its latency analysis”. *arXiv preprint arXiv:1808.03949*, 2018.

94. J. Kim, G. Kim, and B. Han. "Multi-Level Branched Regularization for Federated Learning". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 11058–11073.

95. D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*, 2014.

96. J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik. "Federated optimization: Distributed machine learning for on-device intelligence". *arXiv preprint arXiv:1610.02527*, 2016.

97. J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. "Federated learning: Strategies for improving communication efficiency". *arXiv preprint arXiv:1610.05492*, 2016.

98. J. Kwon, J. Kim, H. Park, and I. K. Choi. "Asam: Adaptive Sharpness-aware Minimization for Scale-invariant Learning of Deep Neural Networks". In: *Proc. of International Conference on Machine Learning*. PMLR. 2021, pp. 5905–5914.

99. Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". *nature* 521:7553, 2015, pp. 436–444.

100. G. Lee, M. Jeong, Y. Shin, S. Bae, and S.-Y. Yun. "Preservation of the Global Knowledge by Not-True Distillation in Federated Learning". In: *Advances in Neural Information Processing Systems*. 2022.

101. G. Legate, L. Caccia, and E. Belilovsky. "Re-Weighted Softmax Cross-Entropy to Control Forgetting in Federated Learning". *arXiv preprint arXiv:2304.05260*, 2023.

102. D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau. "Federated learning for keyword spotting". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 6341–6345.

103. D. Li and J. Wang. "FedMD: Heterogenous Federated Learning via Model Distillation". *arXiv preprint arXiv:1910.03581*, 2019.

104. H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. "Pruning filters for efficient convnets". *arXiv preprint arXiv:1608.08710*, 2016.

105. Q. Li, Y. Diao, Q. Chen, and B. He. “Federated learning on non-iid data silos: An experimental study”. In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE. 2022, pp. 965–978.

106. Q. Li, B. He, and D. Song. “Model-contrastive federated learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10713–10722.

107. S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen. “Abnormal client behavior detection in federated learning”. *arXiv preprint arXiv:1910.09933*, 2019.

108. T. Li, Z. Liu, V. Sekar, and V. Smith. “Privacy for Free: Communication-Efficient Learning with Differential Privacy Using Sketches”. *arXiv preprint arXiv:1911.00972*, 2019.

109. T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. “FedDANE: A Federated Newton-Type Method”. *arXiv preprint arXiv:2001.01920*, 2020.

110. T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. “Federated optimization in heterogeneous networks”. *arXiv preprint arXiv:1812.06127*, 2018.

111. X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. “On the convergence of fedavg on non-iid data”. *arXiv preprint arXiv:1907.02189*, 2019.

112. X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou. “Fedbn: Federated learning on non-iid features via local batch normalization”. *arXiv preprint arXiv:2102.07623*, 2021.

113. X.-C. Li and D.-C. Zhan. “Fedrs: Federated learning with restricted softmax for label distribution non-iid data”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 995–1005.

114. X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. “Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5330–5340.

115. T. Lin, L. Kong, S. U. Stich, and M. Jaggi. "Ensemble distillation for robust model fusion in federated learning". *Advances in Neural Information Processing Systems* 33, 2020, pp. 2351–2363.

116. Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally. "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training". In: *International Conference on Learning Representations*. 2018.

117. Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally. "Deep gradient compression: Reducing the communication bandwidth for distributed training". *arXiv preprint arXiv:1712.01887*, 2017.

118. L. Liu, J. Zhang, S. Song, and K. B. Letaief. "Edge-Assisted Hierarchical Federated Learning with Non-IID Data". *arXiv preprint arXiv:1905.06641*, 2019.

119. M. Liu, H. Jiang, J. Chen, A. Badokhon, X. Wei, and M.-C. Huang. "A collaborative privacy-preserving deep learning system in distributed mobile environment". In: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2016, pp. 192–197.

120. W. Liu, L. Chen, Y. Chen, and W. Zhang. "Accelerating Federated Learning via Momentum Gradient Descent". *IEEE Transactions on Parallel and Distributed Systems* 31:8, 2020, pp. 1754–1766.

121. W. Liu, R. Li, M. Zheng, S. Karanam, Z. Wu, B. Bhanu, R. J. Radke, and O. Camps. "Towards visually explaining variational autoencoders". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8642–8651.

122. Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang. "FedVision: An Online Visual Object Detection Platform Powered by Federated Learning". *arXiv preprint arXiv:2001.06202*, 2020.

123. M. Lukasik, S. Bhojanapalli, A. K. Menon, and S. Kumar. "Teacher's pet: understanding and mitigating biases in distillation". *arXiv preprint arXiv:2106.10494*, 2021.

124. M. Luo, F. Chen, D. Hu, Y. Zhang, J. Liang, and J. Feng. "No fear of heterogeneity: Classifier calibration for federated learning with non-iid data". *Advances in Neural Information Processing Systems* 34, 2021, pp. 5972–5984.

125. L. Lyu, H. Yu, and Q. Yang. "Threats to Federated Learning: A Survey". *arXiv preprint arXiv:2003.02133*, 2020.

126. A. Malekijoo, M. J. Fadaeieslam, H. Malekijou, M. Homayounfar, F. Alizadeh-Shabdiz, and R. Rawassizadeh. "FEDZIP: A Compression Framework for Communication-Efficient Federated Learning". *arXiv preprint arXiv:2102.01593*, 2021.

127. E. K. Markakis, K. Karras, N. Zotos, A. Sideris, T. Moysiadis, A. Corsaro, G. Alexiou, C. Skianis, G. Mastorakis, C. X. Mavromoustakis, and E. Pallis. "EXEGESIS: Extreme Edge Resource Harvesting for a Virtualized Fog Environment". *IEEE Commun. Mag.* 55:7, 2017, pp. 173–179.

128. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. "Communication-efficient learning of deep networks from decentralized data". *arXiv preprint arXiv:1602.05629*, 2016.

129. H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. "Learning differentially private language models without losing accuracy". *arXiv preprint arXiv:1710.06963*, 2017.

130. H. B. McMahan and M. Streeter. "Adaptive bound optimization for online convex optimization". *arXiv preprint arXiv:1002.4908*, 2010.

131. L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. "Exploiting unintended feature leakage in collaborative learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 691–706.

132. M. Mendieta, T. Yang, P. Wang, M. Lee, Z. Ding, and C. Chen. "Local Learning Matters: Rethinking Data Heterogeneity in Federated Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8397–8406.

133. J. Mills, J. Hu, and G. Min. "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT". *IEEE Internet of Things Journal*, 2019.

134. A. Mitani, A. Huang, S. Venugopalan, G. S. Corrado, L. Peng, D. R. Webster, N. Hammel, Y. Liu, and A. V. Varadarajan. "Author Correction: Detection of anaemia from retinal fundus images via deep learning". *Nature Biomedical Engineering* 4:2, 2020, pp. 242–242.

135. A. Mora, D. Fantini, and P. Bellavista. “Federated Learning Algorithms with Heterogeneous Data Distributions: An Empirical Evaluation”. In: *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE. 2022, pp. 336–341.

136. A. Mora, L. Foschini, and P. Bellavista. “Structured Sparse Ternary Compression for Convolutional Layers in Federated Learning”. In: *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*. IEEE. 2022, pp. 1–5.

137. A. Mora, I. Tenison, P. Bellavista, and I. Rish. “Knowledge Distillation for Federated Learning: a Practical Guide”. *arXiv preprint arXiv:2211.04742*, 2022.

138. V. Nair and G. E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

139. A. Narayanan and V. Shmatikov. “Robust de-anonymization of large datasets (how to break anonymity of the Netflix prize dataset)”. *University of Texas at Austin*, 2008.

140. M. Nasr, R. Shokri, and A. Houmansadr. “Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks”. *arXiv preprint arXiv:1812.00910*, 2018.

141. S. Niknam, H. S. Dhillon, and J. H. Reed. “Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges”. *IEEE Communications Magazine* 58:6, 2020, pp. 46–51.

142. T. Nishio and R. Yonetani. “Client selection for federated learning with heterogeneous resources in mobile edge”. In: *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE. 2019, pp. 1–7.

143. K. Niwa, N. Harada, G. Zhang, and W. B. Kleijn. “Edge-consensus Learning: Deep Learning on P2P Networks with Nonhomogeneous Data”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 668–678.

144. J. Oh, S. Kim, and S.-Y. Yun. “Fedbabu: Towards enhanced representation for federated image classification”. *arXiv preprint arXiv:2106.06042*, 2021.

145. T. Panch, P. Szolovits, and R. Atun. “Artificial intelligence, machine learning and health systems”. *Journal of global health* 8:2, 2018.

146. N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar. “Semi-supervised knowledge transfer for deep learning from private training data”. *arXiv preprint arXiv:1610.05755*, 2016.

147. S. R. Pfohl, A. M. Dai, and K. Heller. “Federated and Differentially Private Learning for Electronic Health Records”. *arXiv preprint arXiv:1911.05861*, 2019.

148. L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. “Privacy-preserving deep learning via additively homomorphic encryption”. *IEEE Transactions on Information Forensics and Security* 13:5, 2018, pp. 1333–1345.

149. T. T. Phuong et al. “Privacy-preserving deep learning via weight transmission”. *IEEE Transactions on Information Forensics and Security* 14:11, 2019, pp. 3003–3015.

150. M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar. “Split Learning for collaborative deep learning in healthcare”. *arXiv preprint arXiv:1912.12115*, 2019.

151. X. Qiu, T. Parcollet, D. Beutel, T. Topal, A. Mathur, and N. Lane. “Can Federated Learning Save the Planet?” In: *NeurIPS-Tackling Climate Change with Machine Learning*. 2020.

152. Z. Qu, X. Li, R. Duan, Y. Liu, B. Tang, and Z. Lu. “Generalized Federated Learning via Sharpness Aware Minimization”. In: *Proc. of International Conference on Machine Learning*. PMLR. 2022, pp. 18250–18280.

153. S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays. “Federated Learning for Emoji Prediction in a Mobile Keyboard”. *arXiv preprint arXiv:1906.04329*, 2019.

154. X. Ran, L. Ge, and L. Zhong. “Dynamic margin for federated learning with imbalanced data”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

155. S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan. “Adaptive Federated Optimization”. *arXiv preprint arXiv:2003.00295*, 2020.

156. S. J. Reddi, J. Konečnỳ, P. Richtárik, B. Póczós, and A. Smola. “Aide: Fast and communication efficient distributed optimization”. *arXiv preprint arXiv:1608.06879*, 2016.

157. A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani. “Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization”. *arXiv preprint arXiv:1909.13014*, 2019.

158. A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger. “Braintorrent: A peer-to-peer environment for decentralized federated learning”. *arXiv preprint arXiv:1905.06731*, 2019.

159. F. Sattler, T. Korjakow, R. Rischke, and W. Samek. “Fedaux: Leveraging unlabeled auxiliary data in federated learning”. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

160. F. Sattler, A. Marban, R. Rischke, and W. Samek. “Cfd: Communication-efficient federated distillation via soft-label quantization and delta coding”. *IEEE Transactions on Network Science and Engineering*, 2021.

161. F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. “Robust and communication-efficient federated learning from non-iid data”. *IEEE transactions on neural networks and learning systems*, 2019.

162. F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. “Sparse binary compression: Towards distributed deep learning with minimal communication”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

163. S. Savazzi, M. Nicoli, and V. Rampa. “Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks”. *IEEE Internet of Things Journal*, 2020.

164. R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

165. H. Seo, J. Park, S. Oh, M. Bennis, and S.-L. Kim. “Federated knowledge distillation”. *arXiv preprint arXiv:2011.02367*, 2020.

166. A. Shamir. “How to share a secret”. *Communications of the ACM* 22:11, 1979, pp. 612–613.

167. O. Shamir, N. Srebro, and T. Zhang. “Communication-efficient distributed optimization using an approximate newton-type method”. In: *International conference on machine learning*. 2014, pp. 1000–1008.

168. M. Al-Shedivat, J. Gillenwater, E. Xing, and A. Rostamizadeh. “Federated learning via posterior averaging: A new perspective and practical algorithms”. *arXiv preprint arXiv:2010.05273*, 2020.

169. N. Shoham, T. Avidor, A. Keren, N. Israel, D. Benditkis, L. Mor-Yosef, and I. Zeitak. “Overcoming forgetting in federated learning on non-iid data”. *arXiv preprint arXiv:1910.07796*, 2019.

170. A. Singh, P. Vepakomma, O. Gupta, and R. Raskar. “Detailed comparison of communication efficiency of split learning and federated learning”. *arXiv preprint arXiv:1909.09145*, 2019.

171. V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. “Federated multi-task learning”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4424–4434.

172. J. So, B. Guler, and A. S. Avestimehr. “Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning”. *arXiv preprint arXiv:2002.04156*, 2020.

173. K. Sozinov, V. Vlassov, and S. Girdzijauskas. “Human Activity Recognition Using Federated Learning”. In: *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE. 2018, pp. 1103–1111.

174. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. *The journal of machine learning research* 15:1, 2014, pp. 1929–1958.

175. Statista. *Number of Internet of Things (IoT) Connected Devices Worldwide from 2019 to 2030. 2022.* URL `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`. Accessed on January 2023.

176. N. Strom. "Scalable distributed DNN training using commodity GPU cloud computing". In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

177. G. J. Székely, M. L. Rizzo, N. K. Bakirov, et al. "Measuring and testing dependence by correlation of distances". *The annals of statistics* 35:6, 2007, pp. 2769–2794.

178. A. Z. Tan, H. Yu, L. Cui, and Q. Yang. "Towards personalized federated learning". *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

179. Z. Tang, S. Shi, and X. Chu. "Communication-efficient decentralized learning with sparsification and adaptive peer selection". *arXiv preprint arXiv:2002.09692*, 2020.

180. Z. Tao and Q. Li. "esgd: Communication efficient distributed deep learning on the edge". In: *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*. 2018.

181. I. Tenison, S. A. Sreeramadas, V. Mugunthan, E. Oyallon, E. Belilovsky, and I. Rish. "Gradient masked averaging for federated learning". *arXiv preprint arXiv:2201.11986*, 2022.

182. C. Thapa, M. Chamikara, and S. Camtepe. "SplitFed: When Federated Learning Meets Split Learning". *arXiv preprint arXiv:2004.12088*, 2020.

183. A. Triastcyn and B. Faltings. "Federated Learning with Bayesian Differential Privacy". *arXiv preprint arXiv:1911.10071*, 2019.

184. A. Triastcyn and B. Faltings. "Improved Accounting for Differentially Private Learning". *arXiv preprint arXiv:1901.09697*, 2019.

185. S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. "A hybrid approach to privacy-preserving federated learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 2019, pp. 1–11.

186. F. Varno, M. Saghayi, L. Rafiee, S. Gupta, S. Matwin, and M. Havaei. "Minimizing client drift in federated learning via adaptive bias estimation". *arXiv preprint arXiv:2204.13170*, 2022.

187. P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar. "Reducing leakage in distributed deep learning for sensitive health data". *arXiv preprint arXiv:1812.00564*, 2019.

188. J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, et al. "A field guide to federated optimization". *arXiv preprint arXiv:2107.06917*, 2021.

189. J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor. "Tackling the objective inconsistency problem in heterogeneous federated optimization". *Advances in neural information processing systems* 33, 2020, pp. 7611–7623.

190. S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan. "Adaptive federated learning in resource constrained edge computing systems". *IEEE Journal on Selected Areas in Communications* 37:6, 2019, pp. 1205–1221.

191. Y. Wang, Q. Shi, and T.-H. Chang. "Why Batch Normalization Damage Federated Learning on Non-IID Data?" *arXiv preprint arXiv:2301.02982*, 2023.

192. Wikipedia. *Facebook–Cambridge Analytica data scandal*. URL https://en.wikipedia.org/wiki/FacebookâĂŞCambridge_Analytica_data_scandal. Accessed on May 2020.

193. X. Wu, X. Yao, and C.-L. Wang. "FedSCR: Structure-Based Communication Reduction for Federated Learning". *IEEE Transactions on Parallel and Distributed Systems* 32:7, 2020, pp. 1565–1577.

194. Y. Wu and K. He. "Group Normalization". In: *Proc. of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

195. Z. Wu, S. Sun, Y. Wang, M. Liu, and Q. Liu. "Exploring the Distributed Knowledge Congruence in Proxy-data-free Federated Distillation". *arXiv preprint arXiv:2204.07028*, 2022.

196. C. Xie, S. Koyejo, and I. Gupta. "Asynchronous federated optimization". *arXiv preprint arXiv:1903.03934*, 2019.

197. C. Xie, S. Koyejo, and I. Gupta. "SLSGD: Secure and Efficient Distributed On-device Machine Learning". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2019.

198. W. Xu, W. Fang, Y. Ding, M. Zou, and N. Xiong. "Accelerating Federated Learning for IoT in Big Data Analytics With Pruning, Quantization and Selective Updating". *IEEE Access* 9, 2021, pp. 38457–38466.

199. Y. Xu, X. Qiu, L. Zhou, and X. Huang. "Improving bert fine-tuning via self-ensemble and self-distillation". *arXiv preprint arXiv:2002.10345*, 2020.

200. C. Yang, L. Xie, C. Su, and A. L. Yuille. "Snapshot distillation: Teacher-student optimization in one generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2859–2868.

201. T. Yang, S. Zhu, and C. Chen. "Gradaug: A new regularization method for deep neural networks". *Advances in Neural Information Processing Systems* 33, 2020, pp. 14207–14218.

202. T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays. "Applied federated learning: Improving google keyboard query suggestions". *arXiv preprint arXiv:1812.02903*, 2018.

203. D. Yao, W. Pan, Y. Dai, Y. Wan, X. Ding, H. Jin, Z. Xu, and L. Sun. "Local-Global Knowledge Distillation in Heterogeneous Federated Learning with Non-IID Data". *arXiv preprint arXiv:2107.00051*, 2021.

204. T. Yoon, S. Shin, S. J. Hwang, and E. Yang. "FedMix: Approximation of Mixup under Mean Augmented Federated Learning". In: *International Conference on Learning Representations*. 2020.

205. C.-H. Yu, C.-N. Chou, and E. Chang. "Distributed Layer-Partitioned Training for Privacy-Preserved Deep Learning". In: *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE. 2019, pp. 343–346.

206. Q. Yu, S. Li, N. Raviv, S. M. Mousavi Kalan, M. Soltanolkotabi, and S. Avestimehr. "Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy". In: *International Conference on Artificial Intelligence and Statistics (AISTATS 2019)*. 2019.

207. Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas. “Federated learning based proactive content caching in edge computing”. In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2018, pp. 1–6.

208. M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar. “Adaptive methods for nonconvex optimization”. In: *Advances in neural information processing systems*. 2018, pp. 9793–9803.

209. C. Zhang, P. Patras, and H. Haddadi. “Deep learning in mobile and wireless networking: A survey”. *IEEE Communications Surveys & Tutorials* 21:3, 2019, pp. 2224–2287.

210. H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. “mixup: Beyond empirical risk minimization”. *arXiv preprint arXiv:1710.09412*, 2017.

211. J. Zhang, Z. Li, B. Li, J. Xu, S. Wu, S. Ding, and C. Wu. “Federated learning with label distribution skew via logits calibration”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 26311–26329.

212. J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. “Functional mechanism: regression analysis under differential privacy”. *arXiv preprint arXiv:1208.0219*, 2012.

213. L. Zhang and X. Yuan. “Fedzkt: Zero-shot knowledge transfer towards heterogeneous on-device models in federated learning”. *arXiv preprint arXiv:2109.03775*, 2021.

214. L. Zhang, L. Shen, L. Ding, D. Tao, and L.-Y. Duan. “Fine-tuning global model via data-free knowledge distillation for non-iid federated learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10174–10183.

215. S. Zhang, L. Yao, A. Sun, and Y. Tay. “Deep learning based recommender system: A survey and new perspectives”. *ACM Computing Surveys (CSUR)* 52:1, 2019, pp. 1–38.

216. B. Zhao, K. R. Mopuri, and H. Bilen. “iDLG: Improved Deep Leakage from Gradients”. *arXiv preprint arXiv:2001.02610*, 2020.

217. L. Zhao, S. Hu, Q. Wang, J. Jiang, C. Shen, and X. Luo. "Shielding Collaborative Learning: Mitigating Poisoning Attacks through Client-Side Detection". *arXiv preprint arXiv:1910.13111*, 2019.

218. L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen. "Privacy-preserving collaborative deep learning with unreliable participants". *IEEE Transactions on Information Forensics and Security* 15, 2019, pp. 1486–1500.

219. R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao. "Deep learning and its applications to machine health monitoring". *Mechanical Systems and Signal Processing* 115, 2019, pp. 213–237.

220. Y. Zhao, J. Chen, D. Wu, J. Teng, and S. Yu. "Multi-Task Network Anomaly Detection using Federated Learning". In: *Proceedings of the Tenth International Symposium on Information and Communication Technology*. 2019, pp. 273–279.

221. Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. "Federated learning with non-iid data". *arXiv preprint arXiv:1806.00582*, 2018.

222. J. Zhong, H.-Y. Chen, and W.-L. Chao. "Making Batch Normalization Great in Federated Deep Learning". *arXiv preprint arXiv:2303.06530*, 2023.

223. J. Zhou, Z. Cao, X. Dong, and X. Lin. "PPDM: A privacy-preserving protocol for cloud-assisted e-healthcare systems". *IEEE Journal of Selected Topics in Signal Processing* 9:7, 2015, pp. 1332–1344.

224. T. Zhou and E. Konukoglu. "FedFA: Federated Feature Augmentation". *arXiv preprint arXiv:2301.12995*, 2023.

225. X. Zhou, X. Lei, C. Yang, Y. Shi, X. Zhang, and J. Shi. "Handling Data Heterogeneity in Federated Learning via Knowledge Fusion". *arXiv preprint arXiv:2207.11447*, 2022.

226. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. "Edge intelligence: Paving the last mile of artificial intelligence with edge computing". *Proceedings of the IEEE* 107:8, 2019, pp. 1738–1762.

227. L. Zhu, Z. Liu, and S. Han. "Deep leakage from gradients". In: *Advances in Neural Information Processing Systems*. 2019, pp. 14747–14756.

228. Z. Zhu, J. Hong, and J. Zhou. “Data-free knowledge distillation for heterogeneous federated learning”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12878–12889.