Concepts for Efficient, Adaptive and Robust Deep Learning from Distributed Data

vorgelegt von
M.Sc.
FELIX SATTLER

an der Fakultät IV - Elektrotechnik und Informatik -
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
- Dr. rer. nat. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Henning Sprekeler
1. Gutachter: Prof. Dr. Klaus-Robert Müller
2. Gutachter: Prof. Dr. Thomas Wiegand
3. Gutachter: Prof. Dr. Harold Vincent Poor

Tag der wissenschaftlichen Aussprache: 30. September 2021

Berlin, 2021

Abstract

Concepts for Efficient, Adaptive and Robust Deep Learning from Distributed Data

by Felix SATTLER

Due to their great performance and scalability properties, deep neural networks have become ubiquitous building blocks of many applications. With the rise of mobile and IoT devices, these models now are also being increasingly deployed and trained in distributed settings, where data is heterogeneous and separated by limited communication channels and privacy constraints. These distributed "on-device" training approaches, have many advantages over traditional cloud-based training such as better privacy-preservation, increased security and device autonomy. However, these advantages come at the cost of more challenging training conditions, due to hardware and network constraints of the training environment, statistical heterogeneity of the data, as well as robustness and privacy requirements among others. In this thesis, we present methodologies and algorithmic concepts to address these challenges jointly, by means of heterogeneity-aware and communication-efficient training, robust and adaptive multitask optimization and certainty-weighted aggregation methods. Our proposed solutions reduce communication-overhead in distributed training by up to four orders of magnitude, facilitate personalization and adversarial robustness via automatic device clustering and advance the state of the art in federated training performance in the presence of unlabeled auxiliary data. Our proposed methodologies enable widespread adoption of distributed training solutions, as has been demonstrated through their application to a variety of real-world problems in subsequent work.

Deutsche Zusammenfassung

Tiefe neuronale Netze haben in den vergangenen Jahren beachtliche Durchbrüche erzielt und weite Teile des maschinellen Lernens revolutioniert. Durch ihre Leistungsfähigkeit und Skalierbarkeit sind diese Modelle essentieller Bestandteil vieler Anwendungen geworden. Im Zuge der zunehmenden Verbreitung von Smartphones und intelligenten Geräten kommen tiefe neuronale Netze nun auch vermehrt in Szenarien zur Anwendung, in denen Trainingsdaten verteilt vorliegen und durch statistische Heterogenität gekennzeichnet sind. Aufgrund von Anforderungen an Privatsphäre, Sicherheit und Autonomie ist es in diesen Szenarien oft nicht möglich, die Trainingsdaten an einem zentralen Punkt zu sammeln. Um Daten in dieser Situtation dennoch verwertbar zu machen, muss stattdessen auf verteilte Trainingsprotokolle zurück gegriffen werden, welche die Daten lokal auf jedem Erzeugergerät prozessieren und lediglich abstrakte akkumulierte Trainingsinformationen austauschen. Das verteilte Training birgt jedoch eine Reihe neuer Herausforderungen, welche sich unter anderem aus den Hardware- und Netzwerkbeschränkungen der Trainingsumgebung, der statistischen Heterogenität der Daten, sowie den Robustheits- und Privatsphäreanforderungen ergeben. Die vorliegende Arbeit beschreibt Methoden und Konzepte, welche diese Herausforderungen ganzheitlich addressieren. Die vorgestellten Algorithmen ermöglichen unter anderem eine Reduktion des Datenaustausches im verteilten Training um vier Größenordnungen, erlauben es Geräte automatisiert anhand der Ähnlichkeit ihrer Datenverteilungen zu gruppieren und steigern die Trainingsproduktivität durch adaptive Gewichtung der einzelnen Lerner im Aggregationsprozess. Auf diese Weise werden verteilte Trainingsmethoden auch in der Gegenwart starker Resourcenbeschränkungen zugänglich gemacht, sowie die Personalisierung und Robustheit der gelernten Modelle und des Trainingsprozesses verbessert. Wie durch ihre Verwendung in verschiedenen realen Probleme demonstriert wird, eröffnen die vorgestellten Methoden neue Möglichkeiten bezüglich der Anwendung des verteilten Lernens.

Acknowledgements

I would like to express my gratitude to the people who helped me make this dissertation a reality.

First and foremost I would like to acknowledge my supervisor Prof. Dr. Klaus-Robert Müller for his continued support and encouragement. Prof. Müller always had an open ear, provided me with critical advice and his faith in my ideas helped me push through times of frustration. His miraculous ability to pinpoint holes in any scientific assertion often resulted in extra work for me, which however always paid off in the end.

I also want to thank Dr. Wojciech Samek, Head of the Artificial Intelligence Department at HHI, for being the best mentor, boss and co-author anyone could imagine. Dr. Samek was highly supportive of me from the beginning, endowed me with trust and opened doors by organizing tutorials and special sessions at conferences.

I owe thanks to Prof. Dr. Thomas Wiegand, executive director at HHI, for providing a fantastic research environment and being a driver of motivation during intense project work.

My deep gratitude also goes to Simon Wiedemann, who supervised me during my time as a student research assistant at HHI and initially introduced me to the topic of distributed deep learning. Simon was a great partner in scientific discussions, pointed me at lots of interesting literature and stimulated my imagination far beyond the content of this thesis. Most importantly however, I want to thank Simon for his incredibly positive, cheerful and uplifting attitude at the lab in general and towards me in particular. His optimism and entrepreneurial mindset have had a lasting impression on me.

I also want to thank my colleagues and collaborators at HHI for all the interesting discussions and good fun we had both during and after work. In no particular order: David Neumann, Karsten Müller, Arturo Marban, Tim Korjakow, Haley Hoech, Leyla Arras, Ahmed Osman, Roman Rischke, Sören Becker, Vignesh Srinivasan, Patrick Wagner, Luis Oala and Jackie Ma. Thank you for making our research group not only my intellectual home.

I am also indebted to Gabriela Thiele for taking care of all my administrative needs and always going the extra mile to make my time at HHI as comfortable as possible.

Finally, my immeasurable gratitude goes to Anna, who supported me with her love, admirable patience, and encouragement throughout the entire process of working on my dissertation. When I look at her, holding our newborn son, I am excited for the future and there is nothing that could make me happier.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

CABAC	**C**ontext **A**daptive **B**inary **A**rithmetic **C**oding
CFD	the **C**ompressed **F**ederated **D**istillation algorithm (Sattler et al., 2021a)
CFL	the **C**lustered **F**ederated **L**earning algorithm (Sattler, Müller, and Samek, 2020)
DGC	the **D**eep **G**radient **C**ompression algorithm (Lin et al., 2018)
DNN	**D**eep **N**eural **N**etwork
DSGD	the **D**istributed **S**tochastic **G**radient **D**escent algorithm
FD	**F**ederated **D**istillation
FEDAUX	the **Fed**erated Learning with **A**uxiliary **D**ata algorithm (Sattler et al., 2021b)
FEDAVG	the **Fed**erated **Av**erag**g**ing algorithm (McMahan et al., 2017)
FEDBE	the **Fed**erated Distillation with **B**ayesian **E**nsembles algorithm (Chen and Chao, 2020)
FEDDF	the **Fed**erated **D**istillation for robust model **F**usion algorithm (Lin et al., 2020b)
FEDMD	the **Fed**erated Learning via **M**odel **D**istillation algorithm (Li and Wang, 2019)
FEDPROX	the **Fed**erated Averaging algorithm with **PROX**imal loss term (Li et al., 2020a)
FL	**F**ederated **L**earning
iid	**i**ndependent and **i**dentically **d**istributed
IoT	**I**nternet **o**f **T**hings
KDE	**K**ernel **D**ensity **E**stimation
LSTM	the **L**ong **S**hort-**T**erm **M**emory model
ML	**M**achine **L**earning
MPEG	the **M**otion **P**icture **E**xperts **G**roup
NN	**N**eural **N**etwork
QSGD	the **Q**uantized **S**tochastic **G**radient **D**escent algorithm (Alistarh et al., 2017)
SBC	the **S**parse **B**inary **C**ompression algorithm (Sattler et al., 2019)
SGD	the **S**tochastic **G**radient **D**escent algorithm
signSGD	the **sign**-compressed **S**tochastic **G**radient **D**escent algorithm (Bernstein et al., 2018)
STC	the **S**parse **T**ernary **C**ompression algorithm (Sattler et al., 2020b)
SVD	**S**ingular **V**alue **D**ecomposition
TernGrad	the **Tern**ary **Grad**ient Compression algorithm (Wen et al., 2017)

Für Anna und Elias.

Chapter 1

Introduction

As Internet of Things (IoT) applications raise in popularity and smartphones have become ubiquitous companions in our everyday lifes, the number of intelligent devices in the world has rapidly grown over the last couple of years. Many of these devices are equipped with various sensors and increasingly potent hardware that allow them to collect and process data at unprecedented scales (Taylor, Baron, and Schmidt, 2015).

In a concurrent development, deep learning has revolutionized the ways that patterns and actionable insights can be extracted from data resources with groundbreaking successes in areas such as computer vision, natural language processing or voice recognition among many others (LeCun, Bengio, and Hinton, 2015; Karpathy and Fei-Fei, 2015; Bosse et al., 2018; Karpathy et al., 2014; Sutskever, Vinyals, and Le, 2014; Samek, Wiegand, and Müller, 2018). Deep learning scales well with growing amounts of data and it's astounding successes in recent times can be at least partly attributed to the availability of large and diverse data sets for training. It is self-evident, that there lays huge potential in harnessing the rich data provided by mobile and IoT devices for the training and improving of deep learning models (McMahan et al., 2017).

At the same time however, we see that data privacy has become a growing concern for many users. Multiple cases of data leakage and misuse in recent times have demonstrated that the centralized processing of data comes at a high risk for the end users privacy (McLeod and Dolezel, 2018). As IoT and mobile devices usually collect data in private environments, often even without explicit awareness of the users, these concerns hold particularly strong. In many situations it is therefore not an option to share locally generated data, such as private pictures or text massages, with a centralized data silo in the "cloud" to conduct training of a deep learning model. This leaves us facing the following dilemma: How are we going to make use of the rich combined data of millions of mobile and IoT devices for training deep learning models if this data can not be stored at a centralized location?

Distributed training frameworks, such as Federated Learning (McMahan et al., 2017) or peer-to-peer learning (Bellet et al., 2018), resolve this issue by allowing multiple parties to jointly train deep learning models on their combined data, without any of the participants having to reveal their data to a centralized entity. This form of privacy-preserving collaborative learning is achieved by processing data locally, on-device, and only communicating abstract training information, such as gradients, higher order model updates or soft-label predictions, to a centralized cloud server or other devices directly.

By repeatedly alternating between local training and aggregation of the local client contributions, knowledge diffuses indirectly between the participants and powerful models can be trained that capture patterns on the combined data of all

clients, without this data ever having to leave any of the local devices, ensuring improved privacy, security and attribution of ownership.

Although distributed training systems have the potential to make entirely new realms of data accessible for use in data science pipelines, their practical implementations face many challenges that arise from the often limited and heterogeneous hardware of the participating devices, the statistical heterogeneity in their data as well as the distributed nature of the optimization process itself. Distributed training applications for instance may suffer from constrained communication channels, intermittent connections, device failures as well as malicious participants, which actively try to disturb the joint training effort.

In this thesis, we present methodologies and algorithmic concepts to address these challenges jointly, by means of heterogeneity-aware and communication-efficient training, robust and adaptive multitask optimization and certainty-weighted aggregation methods. Our proposed solutions reduce communication-overhead in distributed training by up to four orders of magnitude, facilitate personalization and adversarial robustness via automatic device clustering and advance the state of the art in federated training performance in the presence of unlabeled auxiliary data.

1.1 Structure of the Thesis

Over the course of this work, we will explore different distributed training scenarios and provide algorithmic and methodological solutions to the challenges that may arise in them. The chapters of this thesis are organized as follows:

Chapter 2 provides an overview of the most important distributed training settings and briefly reviews the dominant algorithmic paradigms for the training of deep neural network classifiers from distributed sources of data. Both opportunities and challenges of distributed training are highlighted and recommendations on simulating distributed training environments are provided.

Chapter 3 explores communication-efficient distributed training in the data center. The communication protocol of sparse k-nary compression is introduced, which provides a novel way of compressing gradient information, by combining sparsification and quantization techniques with efficient encoding and communication delay.

Chapter 4 extends the sparse k-nary compression protocol, introduced in Chapter 3, to the unique challenges of the Federated Learning environment, namely heterogeneous data, partial device participation and bidirectional communication. Theoretical considerations are accompanied by extensive experiments on a wide range of Federated Learning settings.

Chapter 5 investigates communication-efficiency in the context of Federated Distillation. The effects of active distillation-data curation, soft-label quantization and delta-coding techniques on communication and training performance are explored through experiments with large-scale convolutional neural networks and transformer models in heterogeneous Federated Learning environments.

Chapter 6 presents a new way of treating structured data heterogeneity in Federated Learning. Clustered Federated Learning is described as a versatile extension of the conventional Federated Learning framework, which automatically groups clients into clusters of jointly trainable data distributions. Formal clustering criteria are rigorously derived and practical implementation considerations addressed. Clustered Federated

Learning is evaluated in a variety of different Federated Learning settings, including adversarial settings where some clients try to disturb the training process.

Chapter 7 explores Federated Learning in the presence of unlabeled auxiliary data. Certainty-weighted ensemble distillation is described as an extension of conventional Federated Distillation methods for training settings with heterogeneous data. An (ε, δ)-differentially private mechanism to obfuscate the certainty-scores, quantify and limit the privacy loss is presented. Extensive experiments are performed to investigate the influence of different auxiliary data sets and privacy parameters on the performance of our method.

Chapter 8 concludes this thesis with a summary and provides an overview over applications of the concepts and methodologies introduced in this thesis in the sciences as well as an outlook to future work.

1.2 Own Contributions

Conceptual and Technical Contributions:

- Efficient communication protocols for distributed training in the data-center, Federated Learning as well as Federated Distillation are proposed that reduce both upstream and downstream communication by up to four orders of magnitude, while being robust to a wide variety of training conditions (Sattler et al., 2019; Sattler et al., 2020b; Neumann et al., 2019; Sattler et al., 2021a).
- The combination of multiple techniques for communication reduction in distributed training is investigated. Connections between the previously separately treated techniques of communication delay and error accumulation are revealed (Sattler et al., 2019).
- An important practical limitation of conventional Federated Learning, namely structured heterogeneity in the client data distributions, is highlighted (Sattler, Müller, and Samek, 2020).
- The distributed training principle of Clustered Federated Learning (CFL) is proposed (Sattler, Müller, and Samek, 2020), generalizing Federated Learning to settings where the local data distributions exhibit a clustering structure.
- The Clustered Federated Learning framework is investigated w.r.t. several practical concerns, including varying client populations and training with formal privacy guarantees. Tools are proposed that allow CFL to seamlessly adapt to these conditions and constraints (Sattler, Müller, and Samek, 2020).
- Clustered Federated Learning in the presence of malicious or byzantine clients is investigated. It is demonstrated that CFL inherently provides some degree of robustness to these types of adversarial settings (Sattler et al., 2020a).
- FedAUX, a novel certainty-weighted Federated Distillation technique is proposed, which substantially improves performance of Federated Distillation on heterogeneous client data, addressing a long-standing problem in FL research (Sattler et al., 2021b).

Theoretical Contributions:

- A convergence analysis of distributed training algorithms with sparse and quantized parameter updates is provided (Sattler et al., 2020b).
- A computationally efficient and privacy preserving tool, based on the cosine similarity between the clients' gradient updates, is derived, that provably allows to infer the similarity in the data of different members of the client population, thus making it possible to detect clustering structures in local clients data distributions (Sattler, Müller, and Samek, 2020).
- An efficient splitting criterion is derived, which ensures that Clustered Federated Learning only separates the client population if benefits can be expected, making Clustered Federated Learning a flexible general purpose framework for Federated Learning (Sattler, Müller, and Samek, 2020).
- An (ε, δ)-differentially private mechanism to quantify and constrain the privacy loss associated with transmitting certainty scores in weighted Federated Distillation is proposed (Sattler et al., 2021b).
- Certainty-weighted Federated Distillation is analyzed from the perspective of domain adaptation. It is shown that the certainty scores, which are used to weight the client predictions in FedAUX, approximate a robust aggregation rule, proposed in domain adaptation litterature (Sattler et al., 2021b).

Experimental Contributions:

- A thorough analysis of the communication and convergence properties of different Federated Learning methods, as well as their relation to one another, is performed at varying levels of data heterogeneity (Sattler et al., 2019; Sattler et al., 2020b; Sattler et al., 2021a).
- The communication-efficiency and robustness of sparse k-nary compression and Compressed Federated Distillation to a wide variety of Federated Learning settings is demonstrated in extensive experiments on large-scale convolutional neural networks and transformer models.
- Shortcomings of existing efficient Federated Learning protocols are documented in extensive experiments (Sattler et al., 2019; Sattler et al., 2020b; Sattler et al., 2021a).
- The Clustered Federated Learning framework is evaluated on large-scale deep neural networks. Distinctive performance improvements over conventional Federated Learning, when client data exhibits a clustering structure, are demonstrated (Sattler, Müller, and Samek, 2020).
- Federated Learning in the presence of unlabeled auxiliary data is investigated. A wide range of (out-of-distribution) auxiliary data sets are investigated with respect to their suitability for self-supervised pre-training. Drastical performance improvements across different baseline methods are observed when initializing distributed training with pre-trained models obtained in this way (Sattler et al., 2021b).
- The convergence properties of certainty-weighted Federated Distillation are experimentally investigated in hardware-constraint settings (Sattler et al., 2021b).

Contributions not included in this Thesis:

- The investigation of the compression properties of context-adaptive binary arithmetic coding (CABAC) when applied to trained deep neural network representations as well as differential model updates communicated in Federated Learning (Neumann et al., 2019).

1.3 List of Publications

The following list contains all contributions made by the author to the area of deep learning from distributed data. As is common practice in this scientific field, some of the materials presented in this thesis have been pre-published as journal articles or presented at scientific conferences. The work presented in [1-7] and [9] is included in large parts into this thesis. I would like to thank my co-authors for allowing me to use parts of text from previous publications.

Journal articles

[1] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek (2020b). "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9, pp. 3400–3413. DOI: 10.1109/TNNLS.2019.2944481. URL: https://doi.org/10.1109/TNNLS.2019.2944481

[2] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek (2020). "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13. DOI: 10.1109/TNNLS.2020.3015958. URL: https://doi.org/10.1109/TNNLS.2020.3015958

[3] Felix Sattler, Thomas Wiegand, and Wojciech Samek (2020). "Trends and Advancements in Deep Neural Network Communication". In: *ITU Journal: ICT Discoveries* 3.1, pp. 53–63

[4] Felix Sattler, Arturo Marban, Roman Rischke, and Wojciech Samek (2021a). "CFD: Communication-Efficient Federated Distillation via Soft-Label Quantization and Delta Coding". In: *IEEE Transactions on Network Science and Engineering*, pp. 1–1. ISSN: 2327-4697. DOI: 10.1109/TNSE.2021.3081748. URL: https://dx.doi.org/10.1109/TNSE.2021.3081748

Peer-reviewed contributions to conferences

[5] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek (2019). "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication". In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852172. URL: http://dx.doi.org/10.1109/IJCNN.2019.8852172

[6] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek (2019). "Clustered Federated Learning". In: *Proceedings of the NeurIPS'19 Workshop on Federated Learning for Data Privacy and Confidentiality*, pp. 1–5

[7] Felix Sattler, Klaus-Robert Müller, Thomas Wiegand, and Wojciech Samek (2020a). "On the Byzantine Robustness of Clustered Federated Learning". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8861–8865. DOI: 10.1109/ICASSP40776.2020.9054676. URL: http://dx.doi.org/10.1109/ICASSP40776.2020.9054676

[8] David Neumann, Felix Sattler, Heiner Kirchhoffer, Simon Wiedemann, Karsten Müller, Heiko Schwarz, Thomas Wiegand, Detlev Marpe, and Wojciech Samek (2019). "DeepCABAC: Plug&Play Compression of Neural Network Weights and Weight Updates". In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 21–25. DOI: 10.1109/ICIP40778.2020.9190821. URL: https://dx.doi.org/10.1109/ICIP40778.2020.9190821

Preprints

[9] Felix Sattler, Tim Korjakow, Roman Rischke, and Wojciech Samek (2021b). "FedAUX: Leveraging Unlabeled Auxiliary Data in Federated Learning". In: *CoRR* abs/2102.02514. URL: https://arxiv.org/abs/2102.02514

Chapter 2

Deep Learning from Decentralized Data

This Chapter is partly based on

- Felix Sattler, Thomas Wiegand, and Wojciech Samek (2020). “Trends and Advancements in Deep Neural Network Communication”. In: *ITU Journal: ICT Discoveries* 3.1, pp. 53–63

Neural networks have achieved impressive successes in a wide variety of areas of computational intelligence such as computer vision (Hinton et al., 2012; Xu et al., 2015; Karpathy and Fei-Fei, 2015), natural language processing (Bahdanau, Cho, and Bengio, 2015; Kim et al., 2016; Sutskever, Vinyals, and Le, 2014) and speech recognition (Graves and Schmidhuber, 2005) among many others and, as a result, have become a core building block of many applications. As mobile and Internet of things (IoT) devices become ubiquitous parts of our daily lives, neural networks are also being applied in more and more distributed settings. These distributed devices are getting equipped with ever more potent sensors and storage capacities and collect vast amounts of personalized data, which is highly valuable for processing in machine learning pipelines.

When it comes to the processing of data from distributed sources, machine learning in the cloud (*”Cloud ML”*) has been the go-to paradigm in the previous decade (Hwang, 2017). In Cloud ML, local user data is communicated from the often hardware constrained mobile or IoT devices ("clients") to a computationally potent centralized server where it is then processed in a machine learning pipeline. The result of the processing operation (e.g. a trained model or a prediction) may then be sent back to the local device.

From a communication perspective, training schemes which follow the Cloud ML paradigm make use of centralized intelligence and

”Bring the data to the model.”

While the Cloud ML paradigm is convenient for the clients from a computational perspective, as it moves all the workload for processing the data to the computationally potent server, it also has multiple severe drawbacks and limitations, which all arise from the fact that user data is processed at a centralized location:

Privacy: Data collected by mobile or IoT devices is often of private nature and thus bound to the local device. Medical data, text messages, private pictures or footage from surveillance cameras are examples of data which often cannot be processed in the cloud. New data protection legislations like the European GDPR (Voigt and

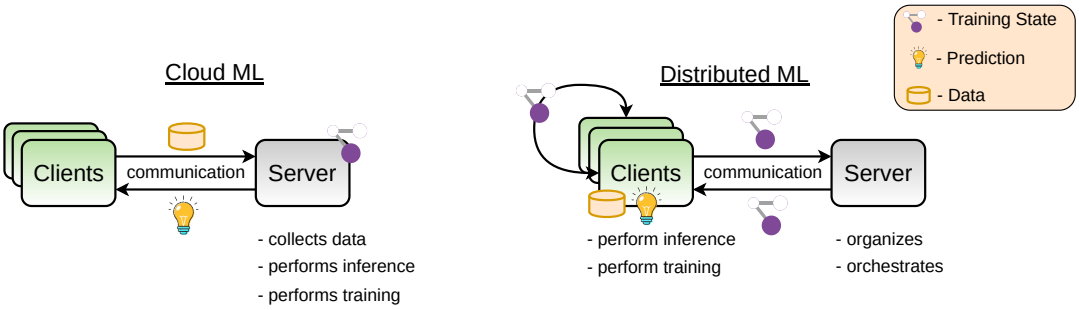


FIGURE 2.1: Comparison between the two paradigms for machine learning from distributed data. In Cloud ML, data from users is collected and processed by a centralized service provider. In Distributed ML, data never leaves the user device. To perform collaborative training, model parametrizations (or equivalent information) are communicated and data is processed locally.

Bussche, 2017) or the Cyber Security Law of the People's Republic of China (Creemers, 2016) enforce strong regulations on data privacy.

Ownership: Attributing and claiming ownership is a difficult task if personal data is transfered to a central location. Cloud ML leaves users in the dark about what happens with their data or requires cumbersome rights management from the cloud service provider.

Security: With all data being stored at one central location, Cloud ML exposes a single point of failure. Multiple cases of data leakage in recent times[1] have demonstrated that the centralized processing of data comes with an unpredictable security risk for the users.

Efficiency: Transferring large records of data to a central compute node often is more expensive in terms of time and energy than the actual processing of the data. For instance, single records of medical image data can already be hundreds of Megabytes in size (Varma, 2012). If the local data is large and/or the communication channels are limited, moving data to the cloud might thus become inefficient or unfeasible.

Autonomy: Many distributed devices need to act fully autonomously and are not allowed to depend on slow and unreliable connections to a cloud server. For instance, in a self-driving car, intelligence responsible for making mission-critical driving decisions needs to be available at all times and thus has to be present on the device.

As awareness for these issues increases and mobile and IoT devices are getting equipped with ever more potent hardware, a new paradigm, termed *"Distributed ML"*, arises with the goal to keep data on device and

"Bring the model to the data."

Multi-party machine learning workflows that follow this paradigm all have one principle in common: In order to avoid the shortcomings of Cloud ML and achieve data locality, they process data on-device and only communicate abstract training information, such as trained neural network models, differential model updates, model gradients or model predictions.

In this chapter, we will provide an overview on machine learning workflows which follow the Distributed ML paradigm, discuss characteristics of the different

[1] A comprehensive list of documented breaches can be found at https://en.wikipedia.org/wiki/List_of_data_breaches. Retrieved June 2021.

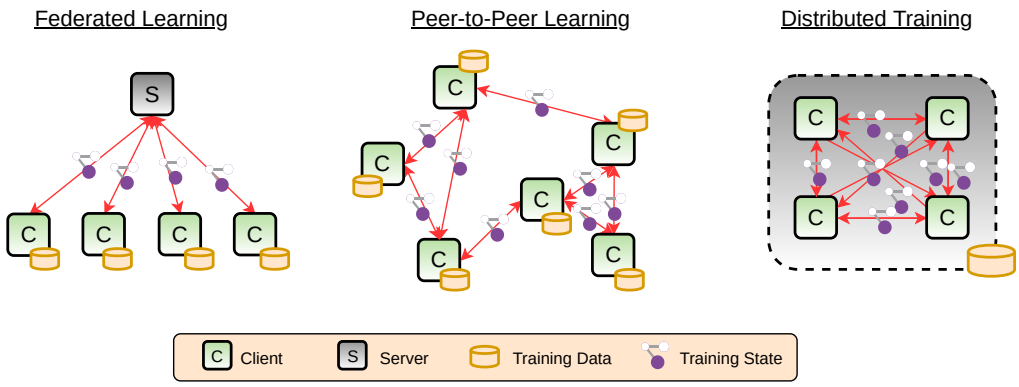


FIGURE 2.2: Communication at the training stages of different Distributed ML pipelines. From left to right: (1) Federated Learning allows multiple clients to jointly train a neural network on their combined data, without any of the local clients having to compromise the privacy of their data. This is achieved by alternating between local optimization and aggregation of model updates at a centralized server. (2) In scenarios where it is undesirable to have a centralized entity coordinating the collaborative training process, peer-to-peer learning offers a potential solution. In peer-to-peer learning the clients directly exchange parameter updates with their neighbors according to some predefined connection topology. (3) In the data center setting, training speed can be drastically increased by splitting the workload among multiple training devices via distributed training. This however requires frequent communication and synchronization of training states between the worker nodes.

setting and briefly review the most popular algorithmic frameworks. Figure 2.1 offers a high level comparison between the two concepts of Cloud ML and Distributed ML.

2.1 Settings

In this section we will review the two most important settings of Distributed ML, namely Federated Learning and peer-to-peer learning. We will also give a short review of distributed training in the data center, as it shares many constraints and challenges with the other two settings and many methods for the compression of neural network representations have been proposed in this domain. All three settings differ with respect to their communication topology, frequency of communication and network constraints. Figure 2.2 illustrates the flow of information in these different settings and Table 2.1 summarizes important characteristics in further detail.

2.1.1 Federated Learning

Federated learning (McMahan et al., 2017; Li, Wen, and He, 2019; Kairouz et al., 2019) is a distributed training setting, which allows multiple parties ("clients") to jointly train machine learning models on their combined data, under the orchestration of a central server.

In Federated Learning, training is performed locally and only parameters, parameter updates or other relevant training information are shared with the central server. Since private data never leaves the local devices, Federated Learning provides strong ownership and security guarantees as well as a basic level of privacy to the participants. Privacy guarantees can be made rigorous by applying cryptographic techniques like homomorphic encryption to the communicated information

TABLE 2.1: Characteristics of different Distributed ML pipelines.

	Distributed Training in the Data Center	Federated Learning	Peer-to-Peer Learning
Setting	high performance computing on a cluster, clients are potent worker nodes with dedicated hardware	privacy preserving training on geographically distributed data, clients are mobile/ IoT devices ("cross device FL") or institutions/ distributed data centers ("cross silo FL")	privacy preserving training on geographically distributed data, updates are exchanged directly between clients without central coordination
Orchestration	the entire training procedure is controlled by a central entity	the joint training effort is orchestrated by a central server, but training is happening locally	no central orchestration, clients train locally, communicate and synchronize directly with each other, organized via a predefined protocol
Data Distribution	can be chosen deliberately, homogeneous and well-behaved	———— data is generated locally and thus may exhibit a **high degree of heterogeneity** ————	
Scale	typically $2-10000$ clients	up to 10^9 clients in the cross-device case, $2-100$ clients in the cross-silo case	up to 10^9 clients
Communication Frequency	high	medium/ low	high
Communication Redundancy	high	medium	high
Communication Objects	model gradients	models/ model updates/ model predictions	gradients/ models/ model updates/ model predictions
Communication Flow	all clients $\rightarrow$ all clients (all-reduce)	some clients $\leftrightarrow$ server	all clients $\rightarrow$ some clients
Communication Channels	dedicated hardwired networks offer high bandwidth and low latency	———— **communication over wireless channels is slow, unreliable and expensive** ————	

(Bonawitz et al., 2017) or by concealing it with differentially private mechanisms (Dwork and Roth, 2014; Geyer, Klein, and Nabi, 2017).

Federated Learning applications can be roughly organized into two categories: In *cross-device* Federated Learning, clients are embodied by hardware constrained mobile or IoT devices. Cross-device Federated Learning settings often are massive in scale. For perspective, consider the example of Google's smart keyboard "Gboard" for which some features are trained with Federated Learning, leveraging a fleet of up to 1.5 Million mobile Android devices (Hard et al., 2018). The limited hardware and communication channels of the participating clients introduce additional challenges in cross-device Federated Learning. For instance, mobile devices may only be capable of participating if they are charging, connected to an unmetered network, and idle. *Cross-silo* Federated Learning on the other hand enables privacy-preserving collaborative training between different institutions or data centers. Cross-silo applications have been proposed in a range of different domains including healthcare informatics

(Xu et al., 2021), pharmaceuticals discovery, electronic health records mining[2], medical data segmentation (Courtiol et al., 2019), and smart manufacturing[3]. Scale and hardware constraints are less of an issue in this setting.

Both in cross-device and in cross-silo Federated Learning applications the training data on a given client is generated based on the specific environment or usage pattern of the mobile device, sensor or local data silo. Therefore the distribution of data among the clients will usually be "non-iid" meaning that any particular client's local data set will not be representative of the whole distribution. The amount of local data will also typically be unbalanced among clients, since different users may make use of their device or a specific application to a different extent. Many scenarios are imaginable in which the total number of clients participating in the optimization is much larger than the average number of training data examples per client. The intrinsic heterogeneity of client data in Federated Learning introduces new challenges when it comes to designing (efficient) training algorithms.

Another characteristic of Federated Learning is the massive communication overhead that arises from keeping clients synchronized. Naively adapting parallel training procedures like distributed stochastic gradient descent to the federated domain, results in massive communication overhead as each participating client has to download and upload a full model gradient during every training iteration. As every such gradient is of the same size as the full model, which can be in the range of gigabytes for modern architectures with millions of parameters, this can quickly overwhelm mobile connections which are often slow, expensive and unreliable.

Federated Learning typically assumes a star-shape communication topology, where all clients directly communicate with the server. In some situations it might however be beneficial to consider also hierarchical communication topologies where the devices are organized at multiple levels. This communication topology naturally arises, for instance, in massively distributed IoT settings, where geographically proximal devices are connected to the same edge server.

2.1.2 Peer-to-Peer Learning

Training with one centralized server might be undesirable in some scenarios, because it introduces a single point of failure and requires the clients to trust a centralized entity (at least to a certain degree). Fully decentralized peer-to-peer learning (Vanhaesebrouck, Bellet, and Tommasi, 2017; Bellet et al., 2018; Lalitha et al., 2019) overcomes these issues, as it allows clients to directly communicate with one another. In this scenario it is usually assumed that the connectivity structure between the clients is given by a connected graph. Given a certain connectivity structure between the clients, peer-to-peer learning is typically realized via a gossip communication protocol, where in each communication round all clients perform one or multiple steps of stochastic gradient descent and then average their locally updated models with those from all their peers. Communication in peer-to-peer learning may thus be high frequent and involve a large number of clients. As clients typically are embodied by mobile or IoT devices which collect local data, peer-to-peer learning shares many properties and constraints of Federated Learning when it comes to the distribution of client data, local processing power and communication bandwidth. In particular, the issues related to non-iid data and slow communication channels discussed above apply in a similar fashion. A unique characteristic of peer-to-peer learning is that there is no central entity which orchestrates the training process. Making decisions about

[2] https://featurecloud.eu/about/our-vision/. Retrieved June 2021.
[3] http://musketeer.eu/project/. Retrieved June 2021.

training related meta parameters may thus require additional consensus mechanisms, which could be realized e.g. via blockchain technology (Chen et al., 2018c). The lack of a single orchestrator also makes it more difficult to handle faulty or malicious clients or to incentivise participation via reward mechanisms. Due to these additional requirements practical application of peer-to-peer learning is currently sparse. Popularity of these methods however could quickly increase once smart-contracts become widely available and accessible.

2.1.3 Distributed Training in the Data Center

Training modern neural network architectures with millions of parameters on huge data sets such as ImageNet can take prohibitively long time, even on the latest high-end hardware. In distributed training in the data center, the computation of stochastic mini-batch gradients is parallelized over multiple machines in order to reduce training time. In order to keep the worker nodes synchronized during this process, they need to communicate their locally communicated gradient updates after every iteration, which results in very high-frequent communication of neural network parameterizations. This communication is time consuming for large neural network architectures and limits the benefits of parallelization according to Amdahl's law (Lewis, 1994). As a result, a large body of research has been devoted to the development of gradient compression techniques, which come with different trade-offs with respect to achievable compression rate, computational overhead of encoding and decoding and suitability for different model aggregation schemes. We will review the most popular techniques in the next chapter. When communication constraints are properly addressed, distributed training applications can scale to up to tens of thousands of workers (Jia et al., 2018) and allow training of multi-billion parameter models in acceptable time (Brown et al., 2020). In contrast to Federated Learning and peer-to-peer learning, in the data center the full data set is accessible to all clients during training. Consequently, the distribution of data among workers can be chosen deliberately and will typically be homogeneous and well-behaved.

The most typical connectivity structure in distributed training in the data center, is an all-to-all connection topology where all computing devices are directly connected via hard-wire. An all-to-all connection allows for efficient model update aggregation via all-reduce operations (Dean and Ghemawat, 2008) and can be easily combined with compression methods as communication is uni-directional.

In this thesis we will cover deep learning from distributed sources of data in the above settings, with a strong focus on Federated Learning and a minor focus on distributed training in the data center. As there is non-trivial overlap between Federated Learning and peer-to-peer learning, many of our proposed methodologies can be generalized to the latter setting. For instance, while not explicitly investigated in this thesis, we believe that the communication-efficient Federated Learning schemes we describe in Chapter 4 and 5 as well as the heterogeneity-robust weighted ensemble distillation method described in Chapter 7 could straight-forwardly be extended to peer-to-peer training applications. We believe that investigations along these lines are an interesting direction of future research.

2.2 Challenges

While distributed training schemes offer a wide range of advantages over conventional Cloud ML solutions, they also come with a unique set of challenges. Despite recent progresses made, many unresolved issues still remain. Some of the most pressing challenges for Distributed ML include:

Data Heterogeneity: As the training data present on the individual clients is collected locally, based on the specific environment and usage pattern, both the size and the distribution of the local data sets will typically vary heavily between different devices. This data generation paradigm violates frequently-used independent and identically distributed ("iid") assumptions in distributed optimization, renders almost all convergence results for distributed learning algorithms inapplicable and has dramatic effects on the practical performance of efficient distributed training algorithms as we will demonstrate in the following chapters.

Systems Heterogeneity: Computational and communication capabilities of the devices in distributed training applications may differ due to variability in hardware (CPU, memory), network connectivity (4G, 5G, wifi), and power (battery level). Distributed training methods have to be able to tolerate heterogeneous hardware, variable execution times and device failures. Systems heterogeneity requirements may also force clients to train model architectures which differ with respect to size and capacity, which poses additional challenges to model aggregation.

Scale: As we have outlined above, distributed training environments often constitute of multiple millions of geographically scattered participants (Bonawitz et al., 2019). Furthermore, as the quality of the collaboratively learned model is determined by the combined available data of all clients, collaborative learning environments will have a natural tendency to grow. This introduces new challenges with respect to device management and optimization.

Limited Resources: Mobile and embedded devices often are not connected to a power grid. Instead their capacity to run computations is limited by a finite battery. Although many research efforts have gone into reducing the complexity of models through neural architecture search (Wu et al., 2019), designing energy-efficient neural network representations (Wiedemann, Müller, and Samek, 2020), or tailoring energy-efficient hardware components (Chen et al., 2016b), the energy efficiency of on-device training is still a big challenge. Consequentially, optimizing large models like deep neural networks with primitives like stochastic gradient descent (SGD) is notoriously expensive, which makes it necessary to keep the number of optimization steps on every client as small as possible. Mobile and embedded devices also typically have only very limited memory. As the memory footprint of SGD grows linearly with the batch size, this might force the devices to train on very small batch sizes.

Privacy: Distributed ML applications promise to preserve the privacy of the local data sets. However, multiple recent works have demonstrated that in adversarial settings information about the training data can be leaked via the parameter updates (Hitaj, Ateniese, and Perez-Cruz, 2017). A combination of cryptographic techniques such as Secure Multi-Party Computation (Goldreich, 1998) and Trusted Execution Environments (Subramanyan et al., 2017), as well a quantifiable privacy guarantees provided by differential privacy (Dwork and Roth, 2014) can help to overcome these issues. However it is still unclear how these techniques can be effectively combined with methods for compressed communication and what optimal trade-offs can be made between performance of the jointly trained model and privacy guarantees for the participants.

TABLE 2.2: Overview of the different challenges in Distributed ML and the chapters in this thesis which address them.

Challenge	Chapter				
	3	4	5	6	7
Data Heterogeneity		✓	✓	✓	✓
Systems Heterogeneity		✓	✓		✓
Scale		✓			✓
Limited Resources	✓	✓	✓		✓
Privacy				✓	✓

Challenge	Chapter				
	3	4	5	6	7
Robustness		✓		✓	
Personalization				✓	
Synchronization		✓	✓		
Partial Participation	✓	✓	✓	✓	✓
Communication	✓	✓	✓		

Robustness: Since privacy guarantees conceal information about the participating clients and their data, there is also an inherent trade-off between privacy and robustness, which needs to be taken into account. For instance, it has been shown that it is possible for an adversary to introduce hidden functionality into the jointly trained model (Bagdasaryan et al., 2020) or disturb the training process (Chen, Su, and Xu, 2017) by communicating adversarially crafted parameter updates. Detecting these malicious behaviors becomes much more difficult under privacy constraints. Methods for distributed training will have to jointly address the issues of efficiency, privacy and robustness.

Personalization: As distributed training applications generally face heterogeneous and unbalanced data available to devices, it may be challenging to ensure good performance across different devices when training a single global model. This often makes it necessary to optimize different models for different client sub-populations or even for each individual client. Inferring relations between the client data in a robust way is a challenging task, especially if other constraints, like privacy, need to be taken into account.

Synchronization: In most distributed learning schemes, communication takes place at regular time intervals such that the state of the system can always be uniquely determined (Chen et al., 2016a). This simplifies theoretical analysis and improves stability of training. However synchronous schemes may also suffer from delayed computation in the presence of slow workers (stragglers). Asynchronous training avoids delays when the time required by workers to compute parameter updates varies heavily. The absence of a central state however makes convergence analysis far more challenging (although convergence guarantees can still be given (De Sa et al., 2015)) and de-synchronization of the workers may cause model updates to become "stale" (Zhang et al., 2016), resulting in a slow down of convergence, especially during the final stages of training.

Partial participation and Device failures: If the number of clients in a distributed learning system is large, full participation of all clients in every training round may lead to diminishing returns in terms of performance improvements on the one hand, and overwhelming cost for model aggregation and synchronization on the other hand. Consequentially it is often more efficient to select only a subset of the total client population for training in every round. This partial participation however makes it more challenging to keep clients synchronized. Furthermore, both in Federated Learning and peer-to-peer learning it can generally not be guaranteed that all clients selected to participate in one particular communication round will report back results. Devices might loose their connection, run out of battery or seize to contribute to the collaborative training for other reasons. These device failures need to be accounted for.

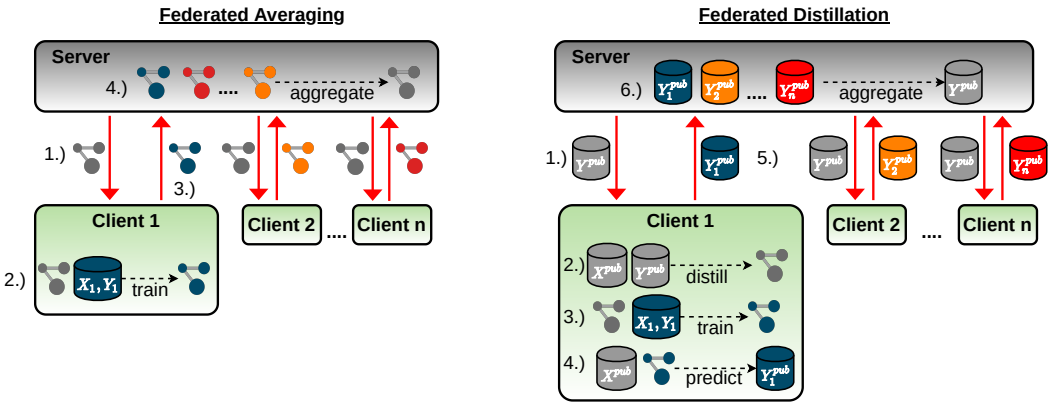


FIGURE 2.3: The flow of data and computations in Federated Averaging and Federated Distillation. In Federated Averaging, the model parameters θ are used to transfer the training information between clients and the server. In Federated Distillation, soft-label predictions Y^{pub} on a common public data set X^{pub} are used to convey the same information.

Communication: A major issue in Distributed ML application is the massive communication overhead that arises from frequently sending around model updates between geographically distributed devices. Client state information, such as model parameters or gradients are of the same size as the fully trained model, which can be in the range of gigabytes for modern architectures with millions of parameters. As communication channels are slow, unreliable and expensive, both the frequency of communication and the size of individual updates need to be reduced drastically to make distributed training feasible. Communicating via a parameter server as is done in Federated Learning introduces additional challenges to communication-efficient distributed training, as now both the training information that is uploaded to the server and the aggregated information that is download from the server need to be compressed in order to reduce communication time and energy consumption.

In this thesis we will jointly address the above challenges. Our proposed algorithmic frameworks and training methodologies will focus in particular on issues of communication-efficiency, data heterogeneity, robustness and personalization, but we will also address privacy concerns and evaluate all of our proposed methods in large-scale settings with partial participation. An overview of the contributions made in the different chapters of this thesis is given in Table 2.2.

2.3 Algorithmic Frameworks

To solve distributed training problems, a number of algorithmic frameworks have been proposed, which differ with respect to their communication properties. In this section we will review the most popular approaches, namely Distributed SGD, Federated Averaging and Federated Distillation. Figure 2.3 gives an overview of these frameworks and compares them w.r.t. to the flow of computation and communication. In the following we will denote the total number of communication rounds by T, the total number of clients by M and the data of each client i by D_i. A high-level procedural comparison between the different frameworks introduced in this section is given in Figure 2.4.

Algorithm 1 Distributed SGD

for $t = 1, .., T$ **do**
  for $i \in \{1, .., M\}$ **do**
    client C_i does:
    sample $D_b^i \subset D_i$
    $g_i \leftarrow \nabla_\theta l(\theta, D_b)$
  end for
  all-reduce on all clients:
  $g \leftarrow \frac{1}{M} \sum_{i=1,..,M} g_i$
  $\theta \leftarrow \theta - \eta g$
end for

Algorithm 2 Federated Averaging

for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., M\}$ **do**
    client C_i does:
    $\theta_i \leftarrow \text{SGD}_E(\theta, D_i)$
  end for
  server S does:
  $\theta \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} \theta_i$
end for

Algorithm 3 Federated Distillation (on server)

for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., M\}$ **do**
    client C_i does:
    $\theta_i \leftarrow \text{SGD}_E(\theta, D_i)$
    $Y_i^{pub} \leftarrow f_{\theta_i}(X^{pub})$
  end for
  server S does:
  $Y^{pub} \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} Y_i^{pub}$
  $\theta \leftarrow \text{SGD}_D(\theta, X^{pub}, Y^{pub})$
end for

FIGURE 2.4: Procedural comparison between the algorithmic frameworks of Distributed SGD, Federated Averaging and Federated Distillation.

2.3.1 Distributed SGD

A popular technique to reduce the training time for large-scale deep neural networks in the data center is to introduce data-parallelism in the computation, by distributing the training data and computation workload evenly among multiple worker nodes (e.g. GPUs) (Chilimbi et al., 2014; Xing et al., 2015; Moritz et al., 2016; Zinkevich et al., 2010). Instead of evaluating a the mini-batch gradient g on a single node, the computation is parallelized according to

$$\sum_{i=1,..,M} g_i = \sum_{i=1,..,M} \sum_{(x,y) \in D_b^i} \nabla_\theta l(x, y, \theta) = \sum_{(x,y) \in \cup_{i=1,..,M} D_b^i} \nabla_\theta l(x, y, \theta) = g. \tag{2.1}$$

After the computation is completed, the clients either send their locally computed gradients to a aggregator node ("server"), or directly exchange and average gradients among each each other via an all-reduce operation. The aggregated gradient is then used to update the master-model by taking a descent step with step-size η to obtain the next iterate on all worker nodes. While this approach, called Distributed Stochastic Gradient Descent (DSGD), is mathematically equivalent to centralized training, easy to implement and can introduce a high level of parallelism into the computation, the delay caused by communicating the gradient information can become a significant bottleneck and slow down the whole distributed learning-system. The problem can become particularly severe if the computation-to-communication ratio is low, for instance when training large-scale recurrent neural networks and transformer models on fast compute nodes. If the training nodes are connected via low-capacity connections, the communication time can dwarf the computation time of forward- and backward-pass and any speed gain from increased parallelism is destroyed. The procedure is summarized in Algorithm 1.

2.3.2 Federated Averaging

The Federated Averaging method (McMahan et al., 2017) was introduced as the algorithmic response to the communication constraints of the Federated Learning setting, namely massive scale, intermittent connections and low-bandwidth channels.

To reduce both the frequency of communication and the total communication load, in Federated Averaging the training is conducted in multiple *communication rounds* following a three step protocol, which consists of the following steps:

1. In the beginning of each communication round $t \leq T$, the central server selects a fraction $C \leq 1$ of the total client population $I_t \subseteq \{1, .., M\}$ and broadcasts to them a common model initialization θ.

2. Starting from that common initialization, the selected clients individually perform a number E steps of stochastic gradient descent, optimizing over their local data D_i, to improve their local models resulting in an updated model θ_i on every client.

3. The updated models θ_i are then communicated back to the server, where they are aggregated (e.g., by an averaging operation) to create a global model θ, which is used as initialization point for the next communication round.

By sub-sampling from the client population and delaying the synchronization of clients for multiple steps of local training, both the total amount of communication and the communication frequency for each client can be reduced by a factor of EC^{-1}. However, every communication round of Federated Averaging still involves the upstream and downstream communication of a complete parametrization of the jointly trained model θ between all participating clients and the server. For large neural networks (like for instance the popular ResNet) which contain multiple millions to billions of individual parameters, the amount of communication may still be prohibitive. As generally both theoretical (Kidger and Lyons, 2020; Chong, 2020) and empirical (Huang et al., 2019) evidence suggests that the performance of neural network models correlates positively with their size, we can not hope that faster mobile networks will remedy this issue in the foreseeable future.

Although a wide variety of methods to reduce the communication overhead in Federated Averaging have been proposed, including approaches that use neural network pruning (LeCun, Denker, and Solla, 1990), message sparsification (Aji and Heafield, 2017; Lin et al., 2018) and other lossy (Courbariaux, Bengio, and David, 2015; Li, Zhang, and Liu, 2016; Konecný et al., 2016; Xu et al., 2020) and loss-less compression techniques (Neumann et al., 2019; Wiedemann et al., 2020), the fundamental issue of scaling to larger models persists.

Moreover, as we will demonstrate in Chapter 4 the performance of Federated Averaging (and many other communication-efficient distributed training algorithms) suffers severely if client data is statistically heterogeneous, as can be expected in Federated Learning applications. To rectify this issue, in this thesis we will propose a novel communication-efficient Federated Learning method, which as we will demonstrate is highly robust to non-iid data.

Federated Averaging also assumes that one single model can be trained on the data of all clients. This assumption however is frequently violated in real Federated Learning applications, especially given the fact that in Federated Learning clients, can hold arbitrary non-iid data, which can not be audited by the centralized server due to privacy constraints. We can imagine for instance scenarios where different preferences of the users lead to a divergent conditional distribution of client data. To address these types of limitations, in Chapter 6 we will introduce a method that automatically separates clients into groups of jointly trainable distribution.

The training procedure of vanilla Federated Averaging is summarized in Algorithm 2.

2.3.3 Federated Distillation

The recently proposed Federated Distillation (Jeong et al., 2018; Lin et al., 2020b; Itahara et al., 2020) takes an entirely different approach to communicating the knowledge obtained during the local training. Instead of communicating the parameterization of the locally trained model θ_i to the server, in Federated Distillation the knowledge is communicated in the form of soft-label predictions on records of a public distillation data set X^{pub} according to

$$Y_i^{pub} = \{f_{\theta_i}(x) | x \in X^{pub}\} \tag{2.2}$$

where f_{θ_i} is the locally trained (neural network) model parameterized by θ_i. Studies have shown that the public distillation data X^{pub} needs to only roughly follow a similar distribution as the privacy-sensitive client data (Lin et al., 2020b; Sattler et al., 2021b). Hence, a wide variety of data sets may be suitable to pose as distillation data. For instance, in many federated computer vision problems, extremely large image corpora like ImageNet (Deng et al., 2009) are publicly available. Likewise, for natural language processing problems, public text corpora like WiKiText (Merity et al., 2017) can be found. While this public data is typically unfit for training a task-specific model due to missing label information, it can still be useful in Federated Distillation pipelines.

Different variations of Federated Distillation have been proposed that vary w.r.t. their communication properties. We will consider the following version of the Federated Distillation protocol for which each communication round consists of the following five steps:

1. At the beginning of every Federated Distillation round, a subset of the client population is selected for participation and synchronizes with the server by downloading aggregated soft-labels, Y^{pub}, on the public data set.

2. The participating clients update their local models by performing model distillation using the downloaded soft-label information. All stochasticity in the distillation process is controlled via random seeds to ensure that all clients end up with the same distilled model θ.

3. The participating clients improve the distilled model by training on their private local data, resulting in improved models θ_i on every client.

4. Using the locally trained model θ_i, the clients compute soft-labels Y_i^{pub} on the public data and send them to the server.

5. The server aggregates the soft-labels for the next communication round.

Variations of this protocol have been proposed in (Jeong et al., 2018; Li and Wang, 2019) and (Itahara et al., 2020). A slightly modified version of this procedure, where distillation is performed on the server instead of the clients is given in Algorithm 3.

As demonstrated in recent studies (Jeong et al., 2018; Lin et al., 2020b; Itahara et al., 2020), Federated Distillation has several advantages over Federated Averaging: First, as model information is aggregated by means of distillation, Federated Distillation allows the participating clients to train different model architectures. This gives additional flexibility in settings where clients have heterogeneous hardware constraints. Federated Distillation also benefits from increased robustness, as adversarial or malicious clients can not directly influence the parametrization of the jointly trained model (only indirectly via their soft-labels). Furthermore, Federated Distillation has

TABLE 2.3: The parameters used to simulate distributed learning environments in this thesis.

Parameter	Number of Clients	Participation Rate	Data Heterogeneity	Balancedness
Symbol	M	C	A, α	G
Range	$[4, 400]$	$[0.1, 1.0]$	$[1, .., 100]$, resp. $[0.01, 100.0]$	$[0.9, 1.0]$

favorable privacy properties, as in contrast to parameter averaging-based Federated Learning algorithms, it is not directly vulnerable to model inversion attacks (Hitaj, Ateniese, and Perez-Cruz, 2017; Geiping et al., 2020). The most significant advantage, however, arises from the fact that Federated Distillation has a completely different communication profile than Federated Averaging. While the upstream and downstream communication in every round of Federated Averaging scales with the size of the jointly trained neural network as

$$\mathfrak{b} \in \mathcal{O}(|\theta|)) \tag{2.3}$$

in Federated Distillation, communication scales with the product of the distillation data set size $|X^{pub}|$ and the number of different classes $\dim(\mathcal{Y})$ as

$$\mathfrak{b} \in \mathcal{O}(|X^{pub}|\dim(\mathcal{Y})). \tag{2.4}$$

This can put FD at an advantage in applications where large neural networks are trained, as is the case for instance in natural language processing and computer vision tasks. Nevertheless, communication in Federated Distillation can still be prohibitive, especially if large distillation data sets are used. To address this issue, in Chapter 5 we will investigate how to further reduce the communication in Federated Distillation by quantizing the soft-label predictions Y_i^{pub} and reducing the size of the public distillation data set X^{pub}.

Federated Distillation also opens up new ways of treating data heterogeneity in Federated Learning. In Chapter 7 we introduce a novel weighted Federated Distillation technique, which weights client predictions on the distillation data set according to the data-specific certainty of each individual client. As we will demonstrate, this leads to improved model fusion and drastically improves performance in heterogeneous environments.

2.4 Simulating Distributed Learning Environments

Distributed Learning environments can come in all kinds of different shapes and forms. To systematically evaluate the algorithms and training methodologies that we will introduce in this thesis, we simulate a broad range of artificial distributed environments, by varying the following parameters:

Client population size M**:** We simulate client populations that consist of up to 400 different clients. This number is sufficient to cover most distributed training scenarios in the data center and cross-silo Federated Learning settings and can approximate most of the dynamics of large-scale cross-device Federated Learning.

Participation rate C**:** To account for the effects of irregularly available devices, device failures and stragglers, we randomly select a fraction C of the total client population for participation in every round. We vary the participation rate between 10% and 100% to simulate different degrees of client availability.

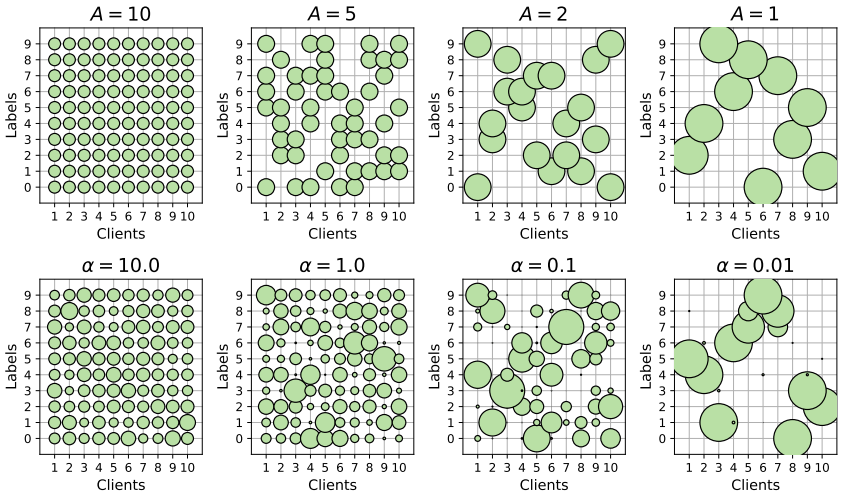


FIGURE 2.5: Illustration of the sharding (top) and Dirichlet data splitting strategies (bottom) used throughout the manuscript, exemplary for a Federated Learning setting with 10 Clients and 10 different classes. Marker size indicates the number of samples held by one client for each particular class. Lower values of A and α lead to more heterogeneous distributions of client data. Figure adapted from (Lin et al., 2020b).

Balancedness of client data G: In practical distributed training environments, the local data sets of different clients may vary heavily in size. To simulate different degrees of unbalancedness we split the data among the clients in a way such that the i-th out of M clients is assigned a fraction

$$\rho(G, i) = 0.1 + 0.9 \frac{G^i}{\sum_{j=1}^{n} G^j} \tag{2.5}$$

of the total data. Lower values of G lead to a more heterogeneous concentration of data. We vary G between 0.9 and 1.0 in our experiments.

Heterogeneity of client data A, α: To simulate the statistically heterogeneous data distributions, which are characteristic for federated and peer-to-peer learning problems, in this thesis we use two different data splitting strategies, which are described below.

Sharding strategy: This strategy was first proposed in (McMahan et al., 2017). We first sort the entire data by label, divide it into single-label shards of equal size, and assign each of the clients an equal number of shards in such a way that it holds an equal number of data points from exactly A different classes. If an exactly symmetrical distribution of data is not possible because the base data set contains an uneven distribution of classes, we allow a minimum number of shards to contain more than one label. For values of A that are lower than the total number of classes c, this splitting strategy results in a pathological non-IID partition of the data, while for $A = c$ the data from different clients will be perfectly balanced.

Dirichlet splitting strategy: This strategy was first proposed in (Hsu, Qi, and Brown, 2019) and allows us to smoothly adapt the level of heterogeneity in the client data via the concentration parameter α. To generate the data split, we sample c vectors

$$p_1, .., p_c \sim \mathrm{Dir}(\alpha), \tag{2.6}$$

where c is the number of classes, from the symmetric M-categorical Dirichlet distribution. For all $p_i \in \mathbb{R}^M_{\geq 0}$ it then holds $\|p_i\|_1 = 1$. The vectors are stacked into a matrix

$$P = [p_1, .., p_c] \in \mathbb{R}^{M,c} \tag{2.7}$$

which is standardized, by repeatedly normalizing the columns and rows. This process converges quickly and is stopped after 1000 iterations. Let n_j be the amount of data points belonging to class j in the training data set. Each client i is then assigned $P_{i,j}n_j$ (non-overlapping) data points from all classes $j = 1, .., c$.

Both data splitting strategies are illustrated in Figure 2.5 exemplary for a distributed training setting with $M = 10$ clients and $c = 10$ different classes. In all our experiments performed throughout this thesis, the data splitting process is controlled by a random seed, to ensure that the different baseline methods are all trained on the same split of data. Table 2.3 gives an overview on the parameters used to simulate distributed learning environments in this thesis.

2.5 Summary

We currently witness a convergence between the areas of machine learning, the Internet of Things and communication technology, which not only leads to the generation of tremendous amounts of highly valuable user data, but also opens up new ways to process this data in a privacy-preserving and decentralized manner. The roll-out of data-intensive 5G and 6G networks (Ding et al., 2017; You et al., 2021) and the steadily improving hardware capabilities of mobile and IoT devices will further accelerate this development, and increase popularity of machine learning from distributed sources of data.

In this chapter we gave a brief overview over the most important distributed training settings and introduced the dominant algorithmic paradigms that have been proposed. We discussed the main advantages that distributed training workflows have over centralized Cloud ML solutions and enumerated the main challenges that currently hinder their wide-spread adoption in real-world applications.

Among other issues, we identified excessive communication cost as one of the major roadblocks for training deep neural networks in a decentralized manner. In the next three chapters, we will address the issue of communication-efficiency for different distributed training scenarios and propose communication-efficient variants of the protocols described in this chapter.

Chapter 3

Communication-Efficient Distributed Training

The training of large-scale deep neural networks can be very time consuming and often requires large amounts of computational resources. As we have learned in the previous chapter, a very popular technique to accelerate the training process, is to harness data paralellism and distribute the load of computing mini-batch gradients over multiple workers via the distributed stochastic gradient descent (DSGD) algorithm. However, in order to keep the distributed training process synchronized, DSGD requires all workers to exchange their locally computed gradient updates during every iteration. Every such update is of the same size as the full model, which can be in the range of gigabytes for modern architectures with millions of parameters (He et al., 2016; Huang et al., 2017). Over the course of multiple hundred thousands of training iterations on big data sets the total communication for every client can easily grow to more than a petabyte. Since communication is also high-frequent in this setting, bandwidth can be a significant bottleneck, leading to diminishing returns from increased paralellization and ultimately making distributed deep learning unproductive.

To address this issue, substantial research has gone into the effort of reducing the amount of communication necessary between the clients via lossy gradient compression schemes. In this chapter, we will introduce a novel compression framework that allows for a drastic reduction of communication cost for distributed training by combining multiple powerful compression techniques.

This Chapter is based on

- Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek (2019). "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication". In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852172. URL: http://dx.doi.org/10.1109/IJCNN.2019.8852172 ©2019 IEEE

3.1 Communication in Distributed Training

For the synchronous distributed training scheme described in Algorithm 1, the total amount of bits communicated by every worker node during training is given by

$$\mathtt{b} \in \mathcal{O}(\underbrace{N_{iter} \times \nu}_{=T} \times \underbrace{|\theta| \times (H(\Delta\theta) + \eta)}_{\text{update size}}) \tag{3.1}$$

where N_{iter} is the total number of training iterations (forward-backward passes), ν is the communication frequency, $|\theta|$ is the size of the model, $H(\Delta\theta)$ is the entropy of the parameter updates exchanged during upload and download respectively, and η is the inefficiency of the encoding, i.e. the difference between the true update size and the minimal update size (which is given by the entropy). If we assume the size of the model and number of training iterations to be fixed (e.g. because we want to achieve a certain accuracy on a given task), this leaves us with three options to reduce communication: We can a) reduce the communication frequency ν, b) reduce the entropy of the weight updates $H(\Delta\theta^{up/down})$ via lossy compression schemes and/or c) use more efficient encodings to communicate the weight updates, thus reducing η. A wide variety of compression methods have been proposed to reduce the amount of communication during the training process. Using equation (3.1) as a reference, we can organize the existing research body on communication-efficient distributed deep learning into three different groups:

Communication delay methods reduce the communication frequency ν. The most prominent member of this family of algorithms is the Federated Averaging algorithm (McMahan et al., 2017) introduced in the previous chapter, where instead of communicating after every iteration, the workers perform multiple local iterations of SGD to compute a weight update. The authors empirically observe on different convolutional and recurrent neural network architectures that communication can be delayed for up to 100 iterations without significantly affecting the convergence speed as long as the data is distributed among the clients in an iid manner (as is usually the case in data center applications). The amount of communication can be reduced even further with longer delay periods, however this comes at the cost of an increased number of gradient evaluations. In a follow-up work (Konecný et al., 2016) combine this communication delay with random sparsification and probabilistic quantization. They restrict the clients to learn random sparse weight updates or force random sparsity on them afterwards ("structured" vs "sketched" updates) and combine this sparsification with probabilistic quantization. Their method however significantly slows down convergence speed in terms of SGD iterations.

Sparsification methods reduce the entropy $H(\Delta\theta)$ of the updates by restricting changes to only a small subset of the parameters. Strom (Strom, 2015) presents an approach (later modified by (Tsuzuku, Imachi, and Akiba, 2018)) in which only gradients with a magnitude greater than a certain predefined threshold are sent to the server. All other gradients are accumulated in a residual. This method is shown to achieve upstream compression rates of up to 3 orders of magnitude on an acoustic modeling task. In practice however, it is hard to choose appropriate values for the threshold, as it may vary a lot for different architectures and even different layers. To overcome this issue Aji et al. (Aji and Heafield, 2017) instead fix the sparsity *rate* and only communicate the fraction p entries with the biggest magnitude of each gradient, while also collecting all other gradients in a residual. At a sparsity rate of $p = 0.001$ their method only slightly degrades the convergence speed and final accuracy of the trained model. Lin et al. (Lin et al., 2018) present minor modifications to the work of Aji et al. which even close this small performance gap.

Dense quantization methods reduce the entropy $H(\Delta\theta)$ of the weight updates by restricting the update elements to a reduced set of values. Bernstein et al. propose signSGD (Bernstein et al., 2018), a biased compression method with theoretical convergence guarantees on iid data that quantizes every gradient update to it's binary sign, thus reducing the bit size per update by a factor of $\times 32$. signSGD also incorporates download compression by aggregating the binary updates from all clients by means of a majority vote. Other authors propose to stochastically quantize the gradients

during upload in an unbiased way (TernGrad (Wen et al., 2017), QSGD (Alistarh et al., 2017), ATOMO (Wang et al., 2018)). These methods are theoretically appealing, as they inherit the convergence properties of regular DSGD under relatively mild assumptions. However their empirical performance and compression rates do not match those of sparsification methods.

3.2 On the Accumulation of Gradient Information

Communication delay and accumulating sparsification methods as described above already achieve impressive compression rates, however the phenomenon underlying their successes is still only poorly understood.

We present a new information-theoretic perspective that is based on the observation that both of these approaches achieve compression by accumulating gradient information locally before sending it to the server. In the case of communication delay all gradients are accumulated uniformly for a fixed amount of iterations, while in the case of sparsification methods they are accumulated non-uniformly until they exceed some fixed or adaptive threshold. In both cases the rate of compression is proportional to the number of steps that the updates are being delayed on average.

Consider now the optimization path $\Delta\theta_1, .., \Delta\theta_T$ taken by SGD on the loss-surface between some initialization point θ_0 and the model $\theta_T = \theta_0 + \sum_{t=1}^{T} \Delta\theta_t$ trained for T iterations. Following this path, we can model the changes occurring to any individual weight in the network w as a noisy stochastic process via

$$\Delta\theta^t = s^t + n^t, \; t = 1, .., T \tag{3.2}$$

where s^t denotes the deterministic signal (i.e. the true direction of the minimum), while n^t denotes the noise, induced by mini-batch sampling in SGD ("batch noise") and the stochasticity of the learning process itself ("optimization noise", see Fig. 3.1 for an illustration). Motivated by the central limit theorem and empirical investigations by (Bernstein et al., 2018) we can make the simplifying assumption (a) that this noise n^t is normally distributed at every time-step $n^t \sim \mathcal{N}(0, \sigma^2)$ with the variance being constant in time $\mathbb{V}(n^t) = \sigma^2$ for all $t = 1, .., T$. Since the optimization process has the tendency to damp noise as investigated for instance in (LeCun et al., 2012) it is also reasonable to assume (b) that the noise is (negatively) self-correlated. The noise process is then given by $n^1 = N^1$, $n^t = \beta n^{t-1} + N^t$, with N^t normally distributed and

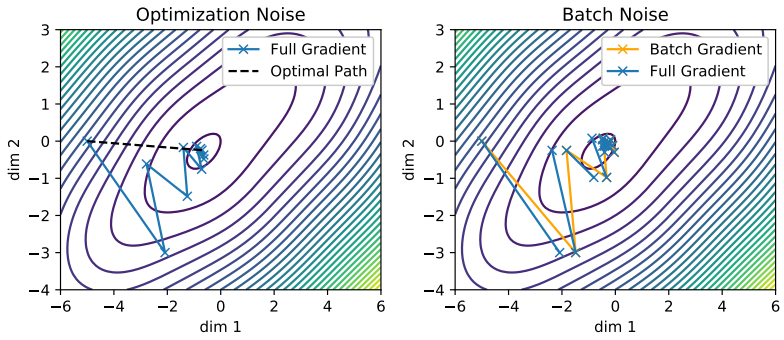


FIGURE 3.1: Sources of noise in SGD (illustration): Left: Optimization noise, caused by Gradient Descent overshooting. Bouncing between the walls of the ravine results in negatively correlated noise. Right: Batch noise, caused by the batch loss being only a noisy approximation of the full empirical loss. ©2019 IEEE

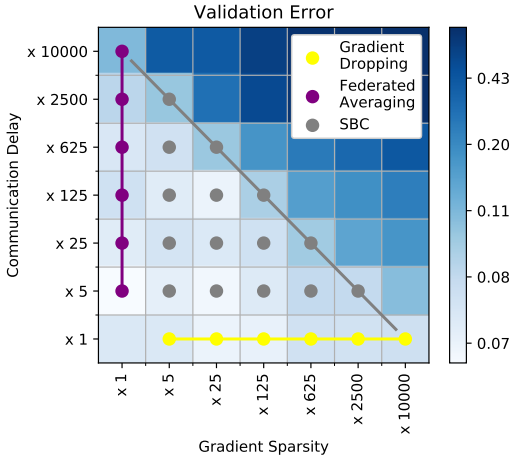


FIGURE 3.2: Validation Error for ResNet32 trained on CIFAR at different levels of temporal and gradient sparsity (the error is color-coded, brighter means lower error). The prior approaches of Gradient Dropping and Federated Averaging can be embedded in a two-dimensional compression framework. ©2019 IEEE

all N^t uncorrelated, $\beta \in (-1, 0)$. Given these assumptions it is straight-forward to bound the variance of the accumulated parameter updates.

Theorem 1. *Under assumptions (a) and (b), the variance of the accumulated noise can be bounded by*

$$\mathbb{V}(\sum_{t=1}^{T} n^t) \leq \sigma^2(T(1+\beta)+1). \tag{3.3}$$

The proof is given in Appendix A.1. Theorem 1 directly leads us to a lower bound on the signal-to-noise ratio of the accumulated weight-updates:

Corollary 1. *Under assumptions (a) and (b), accumulation increases the signal-to-noise ratio from $\bar{s}/\sigma$ to*

$$SNR(\sum_{t=1}^{T} \Delta w^t) := \frac{\mathbb{E}[\sum_{t=1}^{T} s^t + n^t]}{\sqrt{\mathbb{V}[\sum_{t=1}^{T} s^t + n^t]}} \tag{3.4}$$

$$\geq \frac{\sum_{t=1}^{T} s^t}{\sqrt{\sigma^2(T(1+\beta)+1)}} \in \mathcal{O}(\frac{\sqrt{T}}{\sqrt{1+\beta}}\frac{\bar{s}}{\sigma}) \tag{3.5}$$

with $\bar{s} = \frac{1}{T}\sum_{t=1}^{T} s^t$ being the signal-average over time.

This means that a weight-update will be more informative the longer the accumulation period and the stronger the noise correlates temporally. Convergence speed will not be compromised for as long as the information content of the accumulated update is equal to the cumulative information content of the individual updates (c.f. Fig. 3.1 (c)). This line of reasoning helps to shed light on both the successes of communication delay and gradient sparsification.

In fact, it implies that both of these approaches are actually very similar in the way they affect the information flow from client to server on the individual weight level.

We find that this intuition is also verified empirically. Figure 3.2 shows validation errors for ResNet32 model trained on CIFAR for 60000 iterations at different

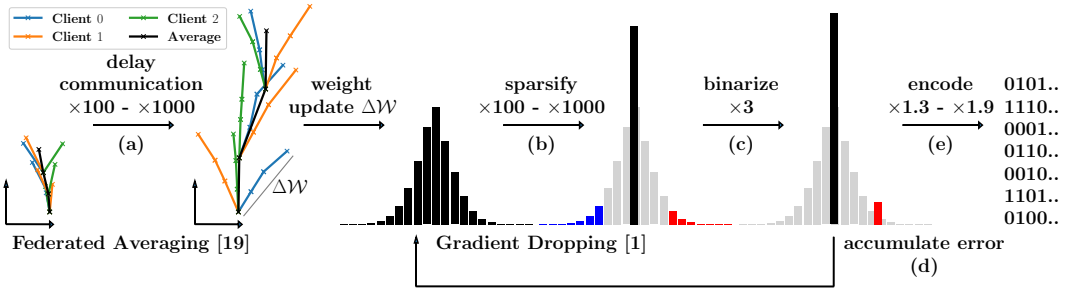


FIGURE 3.3: Step-by-step explanation of techniques used in Sparse Binary Compression: (a) Illustrated is the traversal of the parameter space with regular DSGD (left) and Federated Averaging (right). With this form of communication delay, a bigger region of the loss surface can be traversed, in the same number of communication rounds. That way compression gains of up to $\times 1000$ are possible. After a number of iterations, the clients communicate their locally computed weight updates. (b) Before communication, the weight update is first sparsified, by dropping all but the fraction p weight updates with the highest magnitude. This achieves up to $\times 1000$ compression gain. (c) Then the sparse weight update is binarized for an additional compression gain of approximately $\times 3$. (d) Finally, we optimally encode the positions of the non-zero elements, using Golomb encoding. This reduces the bit size of the compressed weight update by up to another $\times 2$ compared to naive encoding. ©2019 IEEE

levels of communication delay and gradient sparsity. We observe multiple things: a) The validation error remains more or less constant along the off-diagonals of the matrix where the total sparsity (i.e. the product of communication delay and gradient sparsity) is constant. b) The existing methods of Federated Averaging (McMahan et al., 2017) (purple) and Gradient Dropping/ DGC (Aji and Heafield, 2017; Lin et al., 2018)(yellow) are lines in the two-dimensional space of possible compression methods. c) There exists a roughly triangular area of approximately constant error, optimal compression methods lie along the hypotenuse of this triangle. These results indicate, that communication delay and sparsification with error accumulation affect the convergence in a roughly multiplicative way and that there seems to exist a fixed information budged in DSGD, necessary to maintain unhindered convergence.

3.3 Sparse k-nary Compression

Inspired by our findings in the previous section, we propose Sparse k-nary Compression (c.f. Fig. 3.3), to drastically reduce the number of communicated bits in distributed training. SBC makes use of multiple compression techniques simultaneously to reduce all multiplicative components of the total communication (3.1).

Communication Delay, Fig. 3.3 (a): We use communication delay, proposed by (McMahan et al., 2017), to introduce temporal sparsity into DSGD. Instead of communicating gradients after every local iteration, we allow the clients to compute more informative updates by performing multiple iterations of SGD. These generalized weight updates are given by

$$\Delta\theta_i = \mathrm{SGD}_n(\theta_i, D_i) - \theta_i \tag{3.6}$$

where $\mathrm{SGD}_n(\theta_i, D_i)$ refers to the set of weights obtained by performing n iterations of stochastic gradient descent on θ_i, while sampling mini-batches from the i-th client's

training data D_i. For $n = 1$ we obtain regular DSGD.

Sparsification, Fig. 3.3 (b): Inspired by the works of (Lin et al., 2018; Strom, 2015; Shokri and Shmatikov, 2015) and (Aji and Heafield, 2017) we use the magnitude of an individual weight within a weight update as a heuristic for it's importance. Before communication, we set all but the fraction p biggest and fraction p smallest elements of the weight update to zero.

Quantization, Fig. 3.3 (c): The remaining non-zero elements are quantized to further reduce the entropy of each weight update. We experiment with two different quantization strategies:

For our binarization strategy ("Sparse Binary Compression", SBC, Alg. 5), we compute the mean of all remaining positive and all remaining negative elements independently. If the positive mean μ^+ is bigger than the absolute negative mean μ^-, we set all negative values to zero and all positive values to the positive mean and vice versa. For our ternarization strategy ("Sparse Ternary Compression", STC, Alg. 6), we compute the mean μ of the absolute values of all remaining non-zero elements and then set the value of each non-zero element x to $\mu \times \text{sign}(x)$.

Binarizing or ternarizing the non-zero elements of the sparsified weight update to their mean further reduces the entropy of the weight updates by a factor of around $\times 3$. We can get away with quantizing the non-zero weight-update elements because they are relatively homogeneous in value and because we accumulate our compression errors.

Algorithm 4 Sparse k-nary Compression for Efficient Distributed Training

input: initial parameters θ
outout: improved parameters θ
init: all clients C_i are initialized with the same parameters $\theta_i \leftarrow \theta$, the initial global weight update and the residuals are set to zero $\Delta\theta, \mathcal{R}_i \leftarrow 0$

for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., M\}$ **in parallel do**
    Client C_i does:
    $\text{msg} \leftarrow \text{download}_{S \to C_i}(\text{msg})$
    $\Delta\theta \leftarrow \text{golomb\_decode}(\text{msg})$

    $\theta_i \leftarrow \theta_i + \Delta\theta$
    $\Delta\theta_i \leftarrow \mathcal{R}_i + \text{SGD}_n(\theta_i, D_i) - \theta_i$
    $\Delta\theta_i^* \leftarrow \text{sparse\_compression}(\Delta\theta_i, p)$
    $\mathcal{R}_i \leftarrow \Delta\theta_i - \Delta\theta_i^*$

    $\text{msg}_i \leftarrow \text{golomb\_encode}(\Delta\theta_i^*)$
    $\text{upload}_{C_i \to S}(\text{msg}_i)$
  end for
  Server S does:
  $\text{gather}_{C_i \to S}(\Delta\theta_i^*),\ i \in I_t$
  $\Delta\theta \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} \Delta\theta_i^*$
  $\theta \leftarrow \theta + \Delta\theta$
  $\text{broadcast}_{S \to C_i}(\Delta\theta),\ i = 1, .., M$
end for
return θ

Algorithm 5 Sparse Binary Compression

input: flattened tensor $T \in \mathbb{R}^n$, sparsity p
output: sparse binary tensor T^*

$k \leftarrow \max(\lfloor np \rfloor, 1)$
$\mathcal{V}^+ \leftarrow \text{top}_k(T)$; $\mathcal{V}^- \leftarrow \text{top}_k(-T)$
$\mu^+ \leftarrow \text{mean}(\mathcal{V}^+)$; $\mu^- \leftarrow \text{mean}(\mathcal{V}^-)$
if $\mu^+ \geq \mu^-$ **then**
  $T^* \leftarrow \mu^+ \times (T \geq \min(\mathcal{V}^+)$
else
  $T^* \leftarrow -\mu^- \times (T \leq -\min(\mathcal{V}^-)$
end if
return T^*

Algorithm 6 Sparse Ternary Compression

input: flattened tensor $T \in \mathbb{R}^n$, sparsity p
output: sparse ternary tensor T^*

$k \leftarrow \max(\lfloor np \rfloor, 1)$
$\mathcal{V} \leftarrow \text{top}_k(|T|)$
$\mu \leftarrow \text{mean}(\mathcal{V})$
$T^* \leftarrow \mu \times (|T| \geq \min(\mathcal{V})) \odot \text{sign}(T)$
return T^*

Applying sparsification and quantization to an uncompressed weight-update $\Delta\theta_i$ results in a compressed update

$$\Delta\tilde{\theta}_i = \text{sparse\_compression}(\Delta\theta_i) \tag{3.7}$$

Residual Accumulation, Fig. 3.3 (d): It is well established (Lin et al., 2018; Strom, 2015; Aji and Heafield, 2017; Seide et al., 2014; Stich, Cordonnier, and Jaggi, 2018) that the convergence in sparsified DSGD can be greatly accelerated by accumulating the error that arises from only sending sparse approximations of the weight updates. In Section 3.2 we showed that under simplifying assumptions on the SGD noise distribution residual accumulation reduces the SNR of accumulated weight-updates, thus making them more informative. After every communication round τ, the residual is updated via

$$\mathcal{R}_\tau = \sum_{t=1}^{\tau}(\Delta\theta^{(t)} - \Delta\tilde{\theta}^{(t)}) = \mathcal{R}^{(\tau-1)} + \Delta\theta^{(\tau)} - \Delta\tilde{\theta}^{(\tau)}. \tag{3.8}$$

Optimal Position Encoding, Fig. 3.3 (e): To communicate a set of sparse binary/ternary tensors produced by sparse k-nary compression, we only need to transfer the positions of the non-zero elements in the flattened tensors, along with one mean value (μ^+, μ^- or μ) per tensor and one sign-bit in the ternary case. Instead of communicating the absolute non-zero positions it is favorable to only communicate the distances between all non-zero elements. It is possible to show that for big values of $|\theta|$ and $k = p|\theta|$, the distances are approximately geometrically distributed with success probability equal to the sparsity rate p. Therefore, we can optimally encode the distances using the Golomb code (Golomb, 1966). Golomb encoding reduces the average number of position bits to

$$\bar{\mathrm{b}}_{pos} = \mathbf{b}^* + \frac{1}{1-(1-p)^{2^{\mathbf{b}^*}}}, \tag{3.9}$$

with $\mathbf{b}^* = 1 + \lfloor \log_2(\frac{\log(\phi-1)}{\log(1-p)}) \rfloor$ and $\phi = \frac{\sqrt{5}+1}{2}$ being the golden ratio. For a sparsity rate of i.e. $p = 0.01$, we get $\bar{\mathrm{b}}_{pos} = 8.38$, which translates to $\times 1.9$ compression, compared to a naive distance encoding with 16 fixed bits. While the overhead for encoding and decoding makes it unproductive to use Golomb encoding in the situation of (Strom, 2015), this overhead becomes negligible in our situation due to the infrequency of weight update exchange resulting from communication delay.

Our proposed method is described in Algorithms 4, 5 and 6. Algorithm 4 describes how compression and residual accumulation can be introduced into DSGD, while Algorithms 5, 6 describe our sparse ternary and binary compression operators. The algorithms for Golomb encoding and decoding are given in Appendix A.2.

3.4 Convergence Analysis

Using a theoretical framework developed by Stich et al. (Stich, Cordonnier, and Jaggi, 2018) we can prove the convergence of sparse k-nary compression in a simplified setting with $\nu = 1$ and $M = 1$ (no communication delay, only one client) under standard assumptions on the loss function. The proof relies on bounding the impact of the perturbation caused by the compression operator. This is formalized in the following definition:

TABLE 3.1: Final accuracy/perplexity achieved on the test split and average compression rate for different compression schemes in a distributed training setting with four clients on different models and data sets.

Compression Method ⟶		Baseline	DGC [1]	FedAvg [2]	SBC (1)	SBC (2)	SBC (3)
LeNet5-Caffe @MNIST	Accuracy	0.9946	0.994	0.994	0.994	0.994	0.991
	Compression	×1	×718	×500	×2071	×3166	×24935
ResNet18 @CIFAR10	Accuracy	0.946	0.9383	0.9279	0.9422	0.9435	0.9219
	Compression	×1	×768	×1000	×2369	×3491	×31664
ResNet34 @CIFAR100	Accuracy	0.773	0.767	0.7316	0.767	0.7655	0.701
	Compression	×1	×718	×1000	×2370	×3166	×31664
ResNet50 @ImageNet	Accuracy	0.737	0.739	0.724	0.735	0.737	0.728
	Compression	×1	×601	×1000	×2569	×3531	×37208
WordLSTM @PTB	Perplexity	76.02	75.98	76.37	77.73	78.19	77.57
	Compression	×1	×719	×1000	×2371	×3165	×31658
WordLSTM* @WIKI	Perplexity	101.5	102.318	131.51	103.95	103.95	104.62
	Compression	×1	×719	×1000	×2371	×3165	×31657

Definition 1. (Stich, Cordonnier, and Jaggi, 2018) An operator $\text{comp} : \mathbb{R}^d \to \mathbb{R}^d$ is called an α-contraction if it is satisfies the contraction property

$$\mathbb{E}\|x - \text{comp}(x)\|^2 \leq \alpha\|x\|^2, \ \forall x \in \mathbb{R}^d \tag{3.10}$$

for a parameter $0 \leq \alpha < 1$.

We can show that our sparse k-nary compression operators indeed are a α-contraction:

Lemma 2. *Sparse Ternary Compression with sparsity parameter p as defined in Algorithm 6 is a α-contraction, with*

$$\alpha = \frac{\|\text{top}_{pd}(x)\|_1^2}{pd\|x\|_2^2} < 1 \tag{3.11}$$

The proof can be found in Appendix A.3. It then directly follows from Theorem 2.4. in Stich et al. (Stich, Cordonnier, and Jaggi, 2018) that for any L-smooth, ξ-strongly convex objective function l with bounded gradients $\mathbb{E}\|\nabla_\theta l\|^2 \leq U^2$ the averaged updates

$$\overline{\theta_T} = \frac{1}{T}\sum_{t=1,..,T} \theta^{(t)} \tag{3.12}$$

obtained from the update rule

$$\mathcal{R}^{(0)} = 0 \tag{3.13}$$

$$\theta^{(t+1)} = \theta^{(t)} - \text{sparse\_compression}(\mathcal{R}^{(t)} + \eta\nabla_\theta l(\theta^{(t)}), p), \tag{3.14}$$

$$\mathcal{R}^{(t+1)} = \mathcal{R}^{(t)} + \eta\nabla_\theta l(\theta^{(t)}) - \text{sparse\_compression}(\mathcal{R}^{(t)} + \eta\nabla_\theta l(\theta^{(t)}), p) \tag{3.15}$$

converge according to

$$\mathbb{E}[l(\overline{\theta_T})] - l^* \leq \mathcal{O}(\frac{G^2}{\xi T}) + \mathcal{O}(\frac{U^2 L}{p^2\xi T^2}) + \mathcal{O}(\frac{U^2}{p^3\xi T^3}) \tag{3.16}$$

This means that for $T \in \mathcal{O}(p^{-1}(\frac{L}{\xi})^{1/2})$ STC converges at rate $\mathcal{O}(\frac{U^2}{\xi T})$, which is (asymptotically) the same as for regular SGD.

3.5 Experiments

3.5.1 Networks and Data Sets

We evaluate our method on commonly used convolutional and recurrent neural networks with millions of parameters, which we train on well-studied data sets that contain up to multiple millions of samples. We simulate distributed training on four worker nodes for different image classification and language modeling benchmarks.

Image Classification: We run experiments for LeNet5-Caffe[3] on MNIST (LeCun, 1998), ResNet18 and ResNet34 (He et al., 2016) on CIFAR-10 and CIFAR-100 (Krizhevsky, Nair, and Hinton, 2014) and ResNet50 on ILSVRC12 (ImageNet) (Deng et al., 2009). We split the training data homogeneously among the worker nodes, according to the sharding strategy introduced in the previous chapter, with $A = 10$. All models are trained using momentum SGD, except for LeNet5-Caffe, which is trained using the Adam optimizer (Kingma and Ba, 2015). Learning rate, weight intitiallization and data augmentation are as in the respective papers.

Language Modeling: We experiment with multilayer sequence-to-sequence LSTM models as described in (Zaremba, Sutskever, and Vinyals, 2014) on the Penn Treebank (PTB) (Marcus, Marcinkiewicz, and Santorini, 1993) and Wikitext-2 corpora for next-word prediction. The PTB data set consists of a sequence 923000 training, and 82000 validation words, while the Wikitext-2 data set contains 2088628 train and 245569 test words. On both data sets we train a two-layer LSTM model with 650 and 200 hidden units respectively ("WordLSTM" / "WordLSTM*") with tied weights between encoder and decoder as described in (Inan, Khosravi, and Socher, 2017). The training data is split into consecutive subsequences of equal length, out of which we assign one to every client.

While the models we use in our experiments do not fully achieve state-of-the-art results on the respective tasks and data sets, they are still sufficient for the purpose of evaluating our compression method and demonstrate, that our method works well with common regularization techniques such as batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014).

3.5.2 Results

We experiment with three configurations of our method: SBC (1) uses no communication delay and a gradient sparsity of 0.1%, SBC (2) uses 10 iterations of communication delay and 1% gradient sparsity and SBC (3) uses 100 iterations of communication delay and 1% gradient sparsity. All three methods use the binartization mechanism described in Algorithm 5. The experiments with SBC (1) serve the purpose of enabling us to directly compare our 0-value-bit quantization to the 32-value-bit Deep Gradient Compression (Lin et al., 2018)).

Table 3.1 lists compression rates and final validation accuracies achieved by different compression methods, when applied to the training of neural networks on 5 different data sets. The number of iterations (forward-backward-passes) is held constant for all methods. On all benchmarks, our methods perform comparable to the baseline in terms of achieved accuracy, while communicating significantly less bits.

[3] A modified version of LeNet5 from (LeCun et al., 1998).

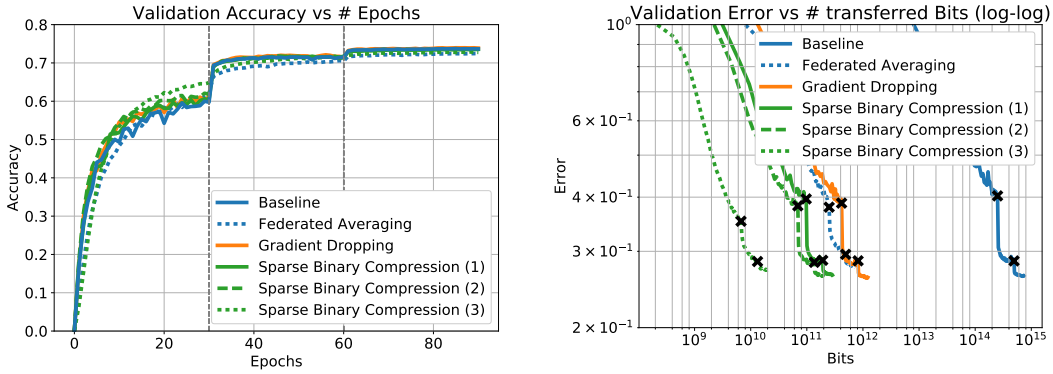


FIGURE 3.4: *Left:* Top-1 validation error vs number of transferred bits (log-log) for ResNet50 trained on ImageNet using different methods for compressed communication. Sparse Binary Compression converges similarly fast as the uncompressed baseline in terms of SGD iterations, while communicating *four orders of magnitude* less bits. *Right:* Convergence speed of different methods for compressed communication for ResNet50 trained on ImageNet. Epochs 30 and 60 at which the learning rate is reduced are marked in the plot. ©2019 IEEE

Figure 3.4 (left) shows convergence speed in terms of iterations for ResNet50 trained on ImageNet. The convergence speed is only marginally affected, by SBC. In the first 30 epochs SBC (3) even achieves the highest accuracy, using about $\times 37000$ less bits than the baseline. In total, SBC (3) reduces the upstream communication on this benchmark from 125 *terabytes* to 3.35 gigabytes for every participating client (c.f. Fig. 3.4 (right)). After the learning rate is lowered in epochs 30 and 60 progress slows down for SBC (3) relative to the methods which do not use communication delay. In direct comparison SBC (1) performs very similar to Gradient Dropping, while using about $\times 4$ less bits (that is $\times 2569$ less bits than the baseline).

3.6 Summary & Limitations

In this chapter we demonstrated that the gradient information communicated when training deep neural networks with SGD is highly redundant. We exploited this fact by combining 3 powerful compression strategies and were able to achieve compression gains of up to four orders of magnitude with only a negligible decrease in accuracy. More fundamentally, we presented theoretical and empirical evidence suggesting that the formerly treated as separate compression methods of communication delay and gradient sparsification with error accumulation in fact can be viewed as two very similar forms of gradient delay that affect the convergence speed in a roughly multiplicative way. Based on this insight we proposed a framework that is able to reap the benefits from both compression approaches and can smoothly adapt to communication-constraints in the learning environment, such as network bandwidth and latency, (SGD-) computation time, as well as temporal inhomogeneities therein.

While our proposed method reliably reduces the amount of communication in distributed neural network training it does so at the cost of additional computational and memory overhead for sparsification, quantization, Golomb encoding and residual bookkeeping. As both the total training time and total energy consumption are determined not only by communication, but also by compute complexity and memory

access, the hyperparameters of our method should to be adjusted to the constraints of the specific hardware setup and model architecture in order to ensure maximum throughput and efficiency.

Lessons Learned

- Weight-updates computed by local learners become more informative, the longer the accumulation period and the stronger the temporal correlation of the optimization noise.
- The popular communication reduction techniques of sparsification with error accumulation and communication delay affect the information flow from clients to server in a similar way on the individual parameter-level.
- A combination of both techniques with quantization and optimal encoding not only improves flexibility in dealing with network constraints, but also yields pareto-superior trade-offs at the communication-performance-frontier.

Chapter 4

Communication-Efficient Federated Learning

In Chapter 3 we introduced the sparse k-nary compression framework and demonstrated that a combination of communication delay, sparsification and quantization of the gradient information can drastically reduce communication overhead in high-performance parallel training applications in the data center. In this chapter, we will extend these ideas to the Federated Learning setting, where communication constraints are even more severe due to the geographical distribution of client devices, their often limited and intermittent access to expensive and low-bandwidth wireless communication network infrastructure as well as their limited energy budget.

While conceptually similar to distributed training in the data-center, Federated Learning comes with a set of additional constraints, which make communication reduction more challenging. These include the heterogeneous distribution of client data, the bi-directional flow of communication between clients and the server as well as the partial participation of client devices and the resulting difficulties in keeping devices synchronized in a communication-efficient way. To apply sparse k-nary compression to the federated setting we will need to address all of these issues.

This Chapter is based on

- Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek (2020b). "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9, pp. 3400–3413. DOI: 10.1109/TNNLS.2019.2944481. URL: https://doi.org/10.1109/TNNLS.2019.2944481

4.1 Limitations of Existing Compression Methods

In this section, we will elaborate on the unique constraints of the Federated Learning environment, which make it challenging to directly apply compression methods that were designed for efficient parallel training in the data center.

4.1.1 Downstream Compression

In distributed training applications in the data center, worker devices are usually hard-wired, physically proximal and low in number. This makes it possible for them to synchronize directly with one-another for instance by means of an all-reduce operation. In Federated Learning, where client populations may contain millions of individual devices, potentially distributed on a global scale, this is not possible. Instead the training process needs to be orchestrated by a central server, which

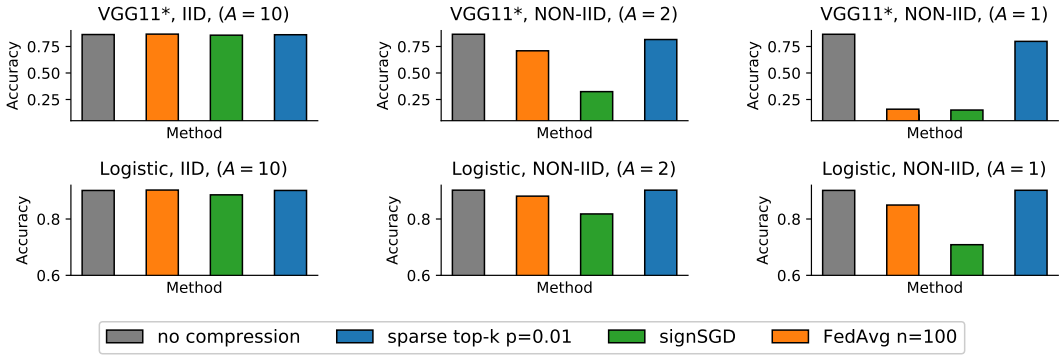


FIGURE 4.1: The effect of data heterogeneity on the performance of different efficient Federated Learning methods. Final accuracy achieved after 100 communication rounds when using different compression methods during the training of VGG11* on CIFAR-10 and Logistic Regression on MNIST in a distributed setting with 10 clients for iid and non-iid data. In the non-iid cases, every client only holds examples from exactly two respectively one of the 10 classes in the data set. All compression methods suffer from degraded convergence speed in the non-iid situation, but sparse top-k is affected by far the least.

aggregates the clients' model updates and keeps the devices synchronized. As a result, communication is bi-directional, with local parameter updates being uploaded to the server and aggregated parameters being communicated back to the clients.

Compression methods, which have been proposed primarily with the intention to speed up parallel training in the data center, thus often can not directly be applied to Federated Learning or loose compression power on the downstream. Sparsification methods for instance will primarily compress the upstream communication, as the sparsity patterns on the updates from different clients will generally differ. If the number of participating clients is greater than the inverse sparsity rate, which can easily be the case in Federated Learning, the aggregated downstream update can become dense, resulting in a loss of compressibility. The same holds true for quantization methods, as the average of quantized parameter updates in general has much higher entropy than the individual updates.

Out of all the communication-efficient methods described in Section 3.1 of the previous chapter, only Federated Averaging and signSGD compress both the upstream and downstream communication.

4.1.2 Partial Participation

The large number of clients in most Federated Learning applications usually makes it impossible for all clients to participate in every round of Federated Learning. This makes it more challenging to apply the efficient quantization and sparsification techniques introduced in the previous chapter to Federated Learning for two reasons. First of all, many of these techniques rely on accumulating quantization errors in a residual. If however clients miss out on a large number of communication rounds, their local residuals may become stale, meaning that the contained gradient signal is no longer useful for the training process and instead will slow down convergence. Secondly, even when no error accumulation is used, partial participation makes it difficult to keep devices synchronized for distributed training algorithms that rely on communicating differential model updates such as gradients or weight-updates from the server to the clients. Assume that an initial model $\theta^{(0)}$ is improved over

T communication rounds by subsequently adding differential update information according to

$$\theta^{(T)} = \theta^{(0)} + \sum_{t=1,..,T} \Delta\theta^{(t)}. \tag{4.1}$$

If a client only participates in a subset of these T communication rounds, it will miss out on all other differential updates leading to a de-synchronization from all other clients, which will harm the convergence.

4.1.3 Robustness to non-iid Data

The related work on efficient distributed deep learning almost exclusively considers iid data distributions among the clients, i.e. they assume unbiasedness of the local gradients with respect to the full-batch gradient according to

$$\mathbb{E}_{x\sim p_i}[\nabla_\theta l(x,\theta)] = \nabla_\theta R(\theta)\ \forall i = 1,..,n \tag{4.2}$$

where p_i is the distribution of data on the i-th client and $R(\theta)$ is the empirical risk function over the combined training data.

While this assumption is reasonable for parallel training where the distribution of data among the clients is chosen by the practitioner, it is typically not valid in the Federated Learning setting where the individual client's gradients will be biased towards the local data set according to

$$\mathbb{E}_{x\sim p_i}[\nabla_\theta l(x,\theta)] = \nabla_\theta R_i(\theta) \neq \nabla_\theta R(\theta)\ \forall i = 1,..,n. \tag{4.3}$$

As it violates assumption (4.2), a non-iid distribution of the local data renders existing convergence guarantees as formulated in (Wen et al., 2017; Alistarh et al., 2017; Bernstein et al., 2018; Wang et al., 2018) inapplicable and has dramatic effects on the practical performance of communication-efficient distributed training algorithms, as we will demonstrate in the following experiments.

We run preliminary experiments with VGG11*, a simplified version of the well-studied 11-layer VGG11 network (Simonyan and Zisserman, 2015), where all dropout and batch normalization layers are removed and the number of convolutional filters and size of all fully-connected layers is reduced by a factor of 2, which we train on the CIFAR-10 (Krizhevsky, Nair, and Hinton, 2014) data set in a Federated Learning setup using 10 clients. To split the training data among the clients, we make use of the sharding strategy introduced in Chapter 2. We also perform experiments with a simple logistic regression classifier, which we train on the MNIST data set (LeCun, 1998) under the same setup of the Federated Learning environment. Both models are trained using momentum SGD. To make the results comparable, all compression methods use the same learning rate and batch size.

Figure 4.1 shows the final accuracy achieved after a fixed number of gradient evaluations for the two models when trained using different methods for communication-efficient Federated Learning. We observe that while all compression methods achieve comparable accuracy on iid data, closely matching the uncompressed baseline, they suffer considerably in the non-iid training settings. As this trend can be observed also for the logistic regression model we can conclude that the underlying phenomenon is not unique to deep neural networks and also carries over to convex objectives. We will now analyze these results in detail for the different compression methods.

TABLE 4.1: Qualitative comparison between different methods for communication-efficient distributed deep learning. None of the existing compression methods satisfies all requirements of the Federated Learning environment.

Method	Downstream Compression	Robust to non-iid Data	Compression Rate
TernGrad (Wen et al., 2017), QSGD (Alistarh et al., 2017), ATOMO (Wang et al., 2018)	✗	✓	$\leq \times 32$
signSGD (Bernstein et al., 2018)	✓	✗	$\times 32$
sparse top-k (Lin et al., 2018; Tsuzuku, Imachi, and Akiba, 2018; Strom, 2015)	✗	✓	$\geq \times 100$
FedAVG (McMahan et al., 2017)	✓	✗	$\geq \times 10$
Sparse k-nary Compression	✓	✓	$\geq \times 100$

✓ = satisfies property, ✗ = does not satisfy property

Federated Averaging: Most noticeably, Federated Averaging (McMahan et al., 2017), although specifically proposed for the Federated Learning setting, suffers considerably from non-iid data. This observation is consistent with Zhao et al. (Zhao et al., 2018) who demonstrated that model accuracy can drop by up to 55% in non-iid learning environments as compared to iid ones. They attribute the loss in accuracy to the increased weight divergence between the clients and propose to side-step the problem by assigning a shared public iid set of labeled data to all clients. While this approach can indeed create more accurate models it also has multiple shortcomings, the most crucial one being that we generally can not assume the availability of such a public data set with label information. If a public data set were to exist one could use it to pre-train a model at the server, which is not consistent with the assumptions typically made in Federated Learning. Furthermore, if all clients share (part of) the same public data set, overfitting to this shared data can become a serious issue. This effect will be particularly severe in highly distributed settings where the number of data points on every client is small. Lastly, even when sharing a relatively large data set between the clients, the original accuracy achieved in the iid situation can not be fully restored. For these reasons, we believe that the data sharing strategy proposed by (Zhao et al., 2018) is an insufficient workaround to the fundamental problem of Federated Averaging having convergence issues on non-iid data.

SignSGD: The quantization method signSGD (Bernstein et al., 2018) suffers from even worse stability issues in the non-iid learning environment. The method completely fails to converge on the CIFAR benchmark and even for the convex logistic regression objective the training plateaus at a substantially degraded accuracy.

Top-k Sparsification: Out of all existing compression methods, top-k sparsification suffers least from non-iid data. For VGG11 on CIFAR the training still converges reliably, even if every client only holds data from exactly one class and for the logistic regression classifier trained on MNIST the convergence does not slow down at all. We hypothesize that this robustness to non-iid data is due to mainly two reasons: First of all, the frequent communication of weight updates between the clients prevents them from diverging too far from one another and hence top-k sparsification does not suffer from weight divergence (Zhao et al., 2018) as it is the case for Federated Averaging. Second, sparsification does not destabilize the training nearly as much as signSGD does since the noise in the stochastic gradients is not amplified by quantization. Although top-k sparsification shows promising performance on non-iid data, it's utility is limited in the Federated Learning setting as it only directly compresses

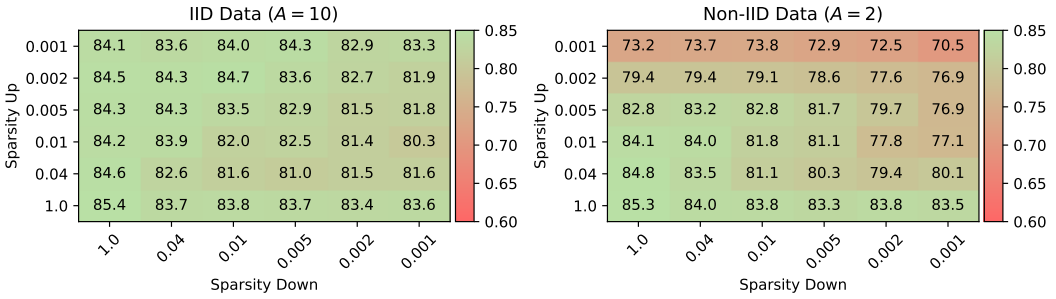


FIGURE 4.2: Accuracy achieved by VGG11* when trained on CIFAR in a distributed setting with 5 clients for 16000 iterations at different levels of upload and download sparsity. Sparsifying the updates for downstream communication reduces the final accuracy by at most 3% when compared to using only upload sparsity.

the upstream communication.

Table 4.1 summarizes our findings: None of the existing compression methods supports both download compression and properly works with non-iid data.

4.2 Applying Sparse k-nary compression to Federated Learning

Top-k sparsification shows the most promising performance in distributed learning environments with non-iid client data. We will use this observation as a starting point to construct an efficient communication protocol for Federated Learning. To extend the communication-efficient sparse k-nary compression protocol to the Federated Learning setting we have to solve three open problems, which prevent the direct application of the method to this new setting:

- First, we will remove the communication delay component by setting $\nu = 1$ as it slows down convergence on heterogeneous data.
- Second, we will incorporate downstream compression into the method to allow for efficient communication from server to clients.
- Third, we will implement a caching mechanism to keep the clients synchronized in case of partial client participation.

4.2.1 Extending to Downstream Compression

Existing compression frameworks which were proposed for distributed training (Lin et al., 2018; Aji and Heafield, 2017; Alistarh et al., 2017; Wen et al., 2017) only compress the communication from clients to server, which is sufficient for applications where aggregation can be achieved via an all-reduce operation. However in the Federated Learning setting, where the clients have to download the aggregated weight-updates from the server this approach is not feasible, as it will lead to a communication bottleneck.

To illustrate this point, let $\mathrm{STC}_k : \mathbb{R}^n \to \mathbb{R}^n, \Delta\theta \mapsto \tilde{\Delta\theta}$ be the compression operator that maps a (flattened) weight update $\Delta\theta$ to a sparsified and ternarized weight update $\tilde{\Delta\theta}$ according to Algorithm 6. For local weight updates $\Delta\theta_i^{(t)}$ the update rule for STC

can then be written as

$$\Delta\theta^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\text{STC}_k(\Delta\theta_i^{(t+1)} + \mathcal{R}_i^{(t)})}_{\Delta\tilde{\theta}_i^{(t+1)}}, \tag{4.4}$$

$$\mathcal{R}_i^{(t+1)} = \mathcal{R}_i^{(t)} + \Delta\theta_i^{(t+1)} - \Delta\tilde{\theta}_i^{(t+1)}, \tag{4.5}$$

starting with an empty residual $\mathcal{R}_i^{(0)} = 0 \in \mathbb{R}^n$ on all clients. While the updates $\Delta\theta_i^{(\tilde{t}+1)}$ that are sent from clients to server are always sparse, the number of non-zero elements in the update $\Delta\theta^{(t+1)}$ that is sent downstream grows linearly with the amount of participating clients in the worst case. If the participation rate exceeds the inverse sparsity $1/p$, the update $\Delta\theta^{(t+1)}$ essentially becomes dense.

To resolve this issue, we propose to apply the same compression mechanism that is used on the clients *also at the server side* to compress the downstream communication. This modifies the update-rule to

$$\Delta\theta^{(\tilde{t}+1)} = \text{STC}_k\Big(\frac{1}{n}\sum_{i=1}^{n} \underbrace{\text{STC}_k(\Delta\theta_i^{(t+1)} + \mathcal{R}_i^{(t)})}_{\Delta\tilde{\theta}_i^{(t+1)}} + \mathcal{R}^{(t)}\Big) \tag{4.6}$$

with a client-side and a server-side residual update

$$\mathcal{R}_i^{(t+1)} = \mathcal{R}_i^{(t)} + \Delta\theta_i^{(t+1)} - \Delta\tilde{\theta}_i^{(t+1)} \tag{4.7}$$

$$\mathcal{R}^{(t+1)} = \mathcal{R}^{(t)} + \Delta\theta^{(t+1)} - \Delta\tilde{\theta}^{(t+1)}. \tag{4.8}$$

We can express this new update rule for both upload and download compression (4.6) as a special case of pure upload compression (4.4) with generalized filter masks: Let M_i, $i = 1, .., n$ be the sparsifying filter masks used by the respective clients during the upload and M be the one used during the download by the server. Then we could arrive at the same sparse update $\Delta\tilde{\theta}^{(t+1)}$ if all clients use filter masks $\tilde{M}_i = M_i \odot M$, where $\odot$ is the Hadamard product. We thus predict that training models using this new update rule should behave similar to regular upstream-only sparsification, but with a slightly increased sparsity rate. We experimentally verify this prediction:

Figure 4.2 shows the accuracies achieved by VGG11 on CIFAR10, when trained in a Federated Learning environment with 5 clients for 10000 iterations at different rates of upload and download compression. As we can see, for as long as download and upload sparsity are of the same order, sparsifying the download is not very harmful to the convergence and decreases the accuracy by at most two percent in both the iid and the non-iid case.

4.2.2 Weight Update Caching for Partial Client Participation

To solve the synchronization problem described in section 4.1.2 and reduce the workload for the clients we propose to use a caching mechanism on the server. Assume the last τ communication rounds have produced the updates $\{\Delta\tilde{\theta}^{(t)} | t = T-1, .., T-\tau\}$. The server can cache all partial sums of these updates up until a certain point $\{P^{(s)} = \sum_{t=1}^{s} \Delta\tilde{\theta}^{(T-t)} | s = 1, .., \tau\}$ together with the global model $\theta^{(T)} = \theta^{(T-\tau-1)} + \sum_{t=1}^{\tau} \Delta\tilde{\theta}^{(T-t)}$. Every client that wants to participate in the next communication round then has to first synchronize itself with the server by either

downloading $P^{(s)}$ or $\theta^{(T)}$, depending on how many previous communication rounds it has skipped. For general sparse updates the bound on the entropy

$$H(P^{(\tau)}) \leq \tau H(P^{(1)}) = \tau H(\tilde{\Delta\theta}^{(T-1)}) \tag{4.9}$$

can be attained. This means that the size of the download will grow linearly with the amount of rounds a client has skipped training. The average number of skipped rounds is equal to the inverse participation fraction $1/C$. This is usually tolerable as the down-link typically is cheaper and has far higher bandwidth than the up-link as already noted in (McMahan et al., 2017) and (Wen et al., 2017). We note that all compression methods, that communicate only parameter updates instead of full models suffer from the same problem. This is also the case for signSGD, although here the size of the downstream update only grows logarithmically with the delay period according to

$$H(P^{(\tau)}_{signSGD}) \leq \log_2(2\tau + 1). \tag{4.10}$$

Partial client participation also has effects on the convergence speed of federated training, both with delayed and sparsified updates. We will investigate these effects in detail in Section 4.3.2.

Algorithm 7 Sparse k-nary Compression for Efficient Federated Learning

input: initial parameters θ
output: improved parameters θ
init: all clients C_i, $i = 1, .., M$ are initialized with the same parameters $\theta_i \leftarrow \theta$, the residuals are initialized to zero $\Delta\theta, \mathcal{R}_i, \mathcal{R} \leftarrow 0$.

for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., M\}$ **in parallel do**
    Client C_i does:
    $\text{msg} \leftarrow \text{download}_{S \rightarrow C_i}(\text{msg})$
    $\Delta\theta \leftarrow \text{golomb\_decode}(\text{msg})$
    $\theta_i \leftarrow \theta_i + \Delta\theta$
    $\Delta\theta_i \leftarrow \mathcal{R}_i + \text{SGD}(\theta_i, D_i, b) - \theta_i$ # Local Training
    $\tilde{\Delta\theta_i} \leftarrow \text{sparse\_compression}(\Delta\theta_i, p_{up})$ # Compression
    $\mathcal{R}_i \leftarrow \Delta\theta_i - \tilde{\Delta\theta_i}$ # Update Residual
    $\text{msg}_i \leftarrow \text{golomb\_encode}(\tilde{\Delta\theta_i})$ # Encoding
    $\text{upload}_{C_i \rightarrow S}(\text{msg}_i)$
  end for
  Server S does:
  $\text{gather}_{C_i \rightarrow S}(\text{msg}_i),\ i \in I_t$
  $\tilde{\Delta\theta_i} \leftarrow \text{golomb\_decode}(\text{msg}_i),\ i \in I_t$
  $\Delta\theta \leftarrow \mathcal{R} + \frac{1}{|I_t|} \sum_{i \in I_t} \tilde{\Delta\theta_i}$ # Aggregation and Residual Update
  $\tilde{\Delta\theta} \leftarrow \text{sparse\_compression}(\Delta\theta, p_{down})$ # Compression
  $\mathcal{R} \leftarrow \Delta\theta - \tilde{\Delta\theta}$
  $\theta \leftarrow \theta + \tilde{\Delta\theta}$ # Server Model Update
  $\text{msg} \leftarrow \text{golomb\_encode}(\tilde{\Delta\theta})$
  $\text{broadcast}_{S \rightarrow C_i}(\text{msg}),\ i = 1, .., M$
end for
return θ

4.3 Experiments

We evaluate our proposed communication protocol on four different learning tasks and compare it's performance to Federated Averaging and signSGD in a wide a variety of different Federated Learning environments.

Models and data sets: To cover a broad spectrum of learning problems we evaluate on differently sized convolutional and recurrent neural networks for the relevant Federated Learning tasks of image classification and speech recognition:

VGG11 on CIFAR*: We train a modified version of the popular 11-layer VGG11 network (Simonyan and Zisserman, 2015) on the CIFAR (Krizhevsky, Nair, and Hinton, 2014) data set. We simplify the VGG11 architecture by reducing the number of convolutional filters to [32, 64, 128, 128, 128, 128, 128, 128] in the respective convolutional layers and reducing the size of the hidden fully-connected layers to 128. We also remove all dropout layers and batch-normalization layers as regularization is no longer required. Batch-normalization has been observed to perform very poorly with both small batch sizes and non-iid data (Ioffe, 2017) and we don't want this effect to obscure the investigated behavior. The resulting VGG11* network still achieves 85.46% accuracy on the validation set after 20000 iterations of training with a constant learning rate of 0.16 and contains 865482 parameters.

CNN on KWS: We train the four-layer convolutional neural network from (Konecný et al., 2016) on the speech commands data set (Warden, 2018). The speech commands data set consists of 51,088 different speech samples of specific keywords. There are 30 different keywords in total and every speech sample is of 1 second duration. Like (Zhao et al., 2018) we restrict us to the subset of 10 most common keywords. For every speech command we extract the mel spectrogram from the short time fourier transform, which results in a 32x32 feature map. The CNN architecture achieves 89.12% accuracy after 10000 training iterations and has 876938 parameters in total.

LSTM on Fashion-MNIST: We also train a LSTM network with 2 hidden layers of size 128 on the Fashion-MNIST data set (Xiao, Rasul, and Vollgraf, 2017). The Fashion-MNIST data set contains 60000 train and 10000 validation greyscale images of 10 different fashion items. Every 28x28 image is treated as a sequence of 28 features of dimensionality 28 and fed as such in the the many-to-one LSTM network. After 20000 training iterations with a learning rate of 0.04 the LSTM model achieves 90.21% accuracy on the validation set. The model contains 216330 parameters.

Compression Methods: We compare our proposed Sparse Ternary Compression method (STC) at a sparsity rate of $p = 1/400$ with Federated Averaging at an "equivalent" delay period of $n = 400$ iterations and signSGD with a coordinate-wise step-size of $\delta = 0.0002$. At a sparsity rate of $p = 1/400$ STC compresses updates both during upload and download by roughly a factor of $\times 1050$. A delay period of $n = 400$ iterations for Federated Averaging results in a slightly smaller compression rate of $\times 400$. During our experiments, we keep all training related hyperparameters constant for the different compression methods. To be able to compare the different methods in a fair way, all methods are given the same budged of training iterations in the following experiments (one communication round of Federated Averaging uses up n iterations, where n is the number of local iterations).

Base Configuration: The Federated Learning environment described in Algorithm 7 can be fully characterized by five parameters: For the base configuration we set the number of clients to 100, the participation ratio to 10% and the local batch size to 20 and assign every client an equally sized subset of the training data containing samples from 10 different classes. In the following experiments, if not explicitly signified otherwise, all hyperparameters will default to this base configuration.

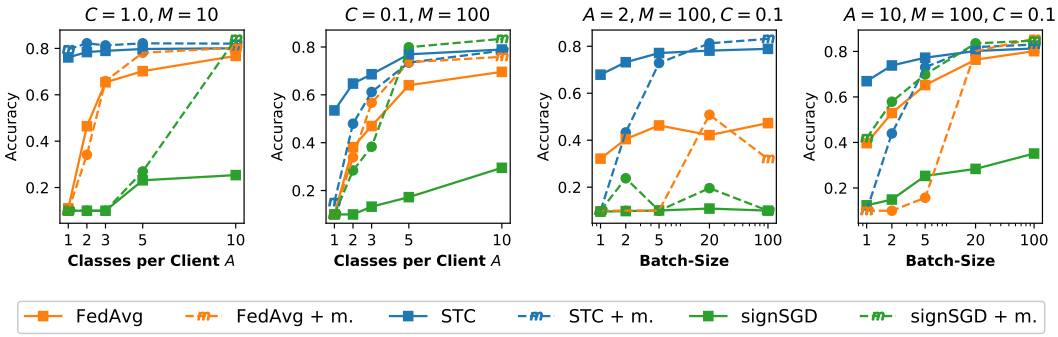


FIGURE 4.3: Maximum accuracy achieved by the different compression methods when training VGG11* on CIFAR for 20000 iterations in varying Federated Learning environments. Left: Robustness of different compression methods to heterogeneity of client data. STC distinctively outperforms Federated Averaging on non-iid data. Right: Robustness of different compression methods to varying batch-sizes. The learning environment is configured as described in Table 2.3. Dashed lines signify that a momentum of $m = 0.9$ was used.

4.3.1 Heterogeneous Client Data

Our preliminary experiments in Section 4.1 have already demonstrated that the convergence behavior of both Federated Averaging and signSGD is very sensitive to the degree of iid-ness of the local client data, whereas sparse communication seems to be more robust. We will now investigate this behavior in some more detail. Figure 4.3 (panel 1, 2) shows the maximum achieved generalization accuracy after a fixed number of iterations for VGG11* trained on CIFAR at different levels of non-iid-ness. Both at full (panel 1) and partial (panel 2) client participation, STC outperforms Federated Averaging across all levels of iid-ness. The most distinct difference can be observed in the non-iid regime, where the individual clients hold less than 5 different classes. Here STC (without momentum) outperforms both Federated Averaging and signSGD by a wide margin. In the extreme case where every client only holds data from exactly one class STC still achieves 79.5% and 53.2% accuracy at full and partial client participation respectively, while both Federated Averaging and signSGD fail to converge at all.

4.3.2 Robustness to other Parameters of the Learning Environment

We will now proceed to investigate the effects of other parameters of the learning environment on the convergence behavior of the different compression methods. Figures 4.3 (panel 3, 4) and 4.4 show the maximum achieved accuracy after training VGG11* on CIFAR for 20000 iterations in different Federated Learning environments.

Local Batch Size: The memory capacity of mobile and IoT devices is typically very limited. As the memory footprint of SGD is proportional to the batch size used during training, clients might be restricted to train on small mini-batches only. Figure 4.3 (panel 3, 4) shows the influence of the local batch size on the performance of different communication-efficient Federated Learning techniques exemplary for VGG11* trained on CIFAR. First of all, we notice that using momentum significantly slows down the convergence speed of both STC and Federated Averaging at batch sizes smaller than 20 independent of the distribution of data among the clients. As we can see, even if the training data is distributed among the clients in an iid manner (Fig. 4.3 panel 4) and all clients participate in every training iteration, Federated Averaging

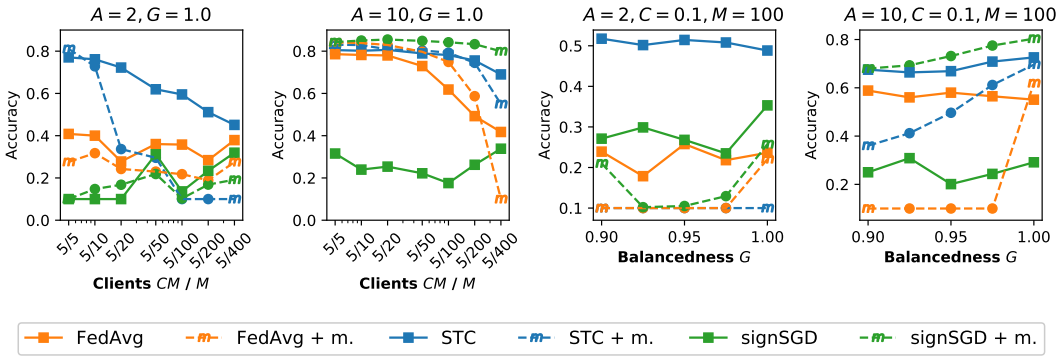


FIGURE 4.4: Left: Robustness of different compression methods to varying client population sizes. The relative client participation fraction is varied between 100% (5/5) and 5% (5/100). Right: Robustness of different compression methods to varying client data set sizes. The training data is split among the client at different degrees of unbalancedness with γ varying between 0.9 and 1.0. Validation accuracy achieved by VGG11* on CIFAR after 20000 iterations

suffers considerably from small batch sizes. STC on the other hand demonstrates to be far more robust to this type of constraint. At an extreme batch size of 1 the model trained with STC still achieves an accuracy of 63.8% while the Federated Averaging model only reaches 39.2% after 20000 training iterations.

Client Participation Fraction: Figure 4.4 (panel 1, 2) shows the convergence speed of VGG11* trained on CIFAR10 in a Federated Learning environment with different degrees of client participation. To isolate the effects of reduced participation, we keep the absolute number of participating clients and the local batch sizes at constant values of 5 and 40 respectively throughout all experiments and vary only the total number of clients (and thus the relative participation C). As we can see, reducing the participation rate has negative effects on both Federated Averaging and STC. The causes for these negative effects however are different: In Federated Averaging the participation rate is proportional to the effective amount of data that the training is conducted on in any individual communication round. If a non-representative subset of clients is selected to participate in a particular communication round of Federated Averaging, this can steer the optimization process away from the minimum and might even cause catastrophic forgetting (Goodfellow et al., 2013) of previously learned concepts. On the other hand, partial participation reduces the convergence speed of STC by causing the clients residuals to go out sync and increasing the gradient staleness (Lin et al., 2018). The more rounds a client has to wait before it is selected to participate during training again, the more outdated it's accumulated gradients become. We can observe this behavior for STC most strongly in the non-iid situation (Fig. 4.4 panel 1), where the accuracy steadily decreases with the participation rate. However even in the extreme case where only 5 out of 400 clients participate in every round of training STC still achieves a higher accuracy than Federated Averaging and signSGD. If the clients hold iid data (Fig. 4.4 panel 2), STC suffers much less from a reduced participation rate than Federated Averaging. If only 5 out of 400 clients participate in every round, STC (without momentum) still manages to achieve an accuracy of 68.2% while Federated Averaging stagnates at 42.3% accuracy. signSGD is affected the least by reduced participation which is unsurprising as only the absolute number of participating clients would have a direct influence on it's performance. Similar behavior can be observed on all other benchmarks, the results can be found in Figure A3 in the appendix. It is noteworthy that in Federated Learning it is usually

TABLE 4.2: Bits required for upload and download to achieve a certain target accuracy on different learning tasks in an iid learning environment. A value of "n.a." in the table signifies that the method has not achieved the target accuracy within the iteration budget. The learning environment is configured as follows: $M = 100, C = 0.1, A = 10, G = 1.0$. The batch-size is set to 20.

	CIFAR, Acc. = 0.84		**KWS, Acc. = 0.9**		**F-MNIST, Acc. = 0.89**	
Method	up	down	up	down	up	down
Baseline	36696.2	36696.2	5191.3	5191.3	2422.8	2422.8
signSGD	1579.5	6937.6	925.1	4063.6	123.3	541.6
FedAvg ($n = 25$)	3572.7	3572.7	301.6	301.6	174.7	174.7
FedAvg ($n = 100$)	1606.3	1606.3	617.3	617.3	83.9	83.9
FedAvg ($n = 400$)	n.a.	n.a.	350.7	350.7	86.5	86.5
STC ($p = 1/25$)	118.4	1184.3	43.5	435.7	8.8	88.4
STC ($p = 1/100$)	202.2	2022.1	31.0	310.2	12.1	121.2
STC ($p = 1/400$)	183.9	1839.7	14.8	148.1	7.9	79.1

possible for the server to exercise some control over the rate of client participation. For instance, it is typically possible to increase the participation ratio at the cost of a longer waiting time for all clients to finish.

Unbalancedness: Up until now, all experiments were performed with a balanced split of data in which every client was assigned the same amount of data points. In practice however, the data sets on different clients will typically vary heavily in size. To simulate different degrees of unbalancedness we vary the parameter G between 0.9 and 1.0, which controls the concentration of data as described in Chapter 2. To amplify the effects of unbalanced client data, we also set the client participation to a low value of only 5 out of 200 clients. Figure 4.4 (panel 3, 4) shows the final accuracy achieved after 20000 iterations for different values of G. Interestingly, the unbalancedness of the data does not seem to have a significant effect on the performance of either of the compression methods. Even if the data is highly concentrated on a few clients (as is the case for $G = 0.9$) all methods converge reliably and for Federated Averaging the accuracy even slightly goes down with increased balancedness. Apparently the rare participation of large clients can balance out several communication rounds with much smaller clients.

4.3.3 Communication-Efficiency

Finally, we compare the different compression methods with respect to the number of iterations and communicated bits they require to achieve a certain target accuracy on a Federated Learning task. As we saw in the previous section, both Federated Averaging and signSGD perform considerably worse if clients hold non-iid data or use small batch sizes. To still have a meaningful comparison we therefore choose to evaluate this time on an iid environment where every client holds 10 different classes and uses a moderate batch size of 20 during training. This setup favors Federated Averaging and signSGD to the maximum degree possible. All other parameters of the learning environment are set to the base configuration. We train until the target accuracy is achieved or a maximum amount of iterations is exceeded and measure the amount of communicated bits both for upload and download. Figure 4.5 shows the results for VGG11* trained on CIFAR, CNN trained on KWS and the LSTM model

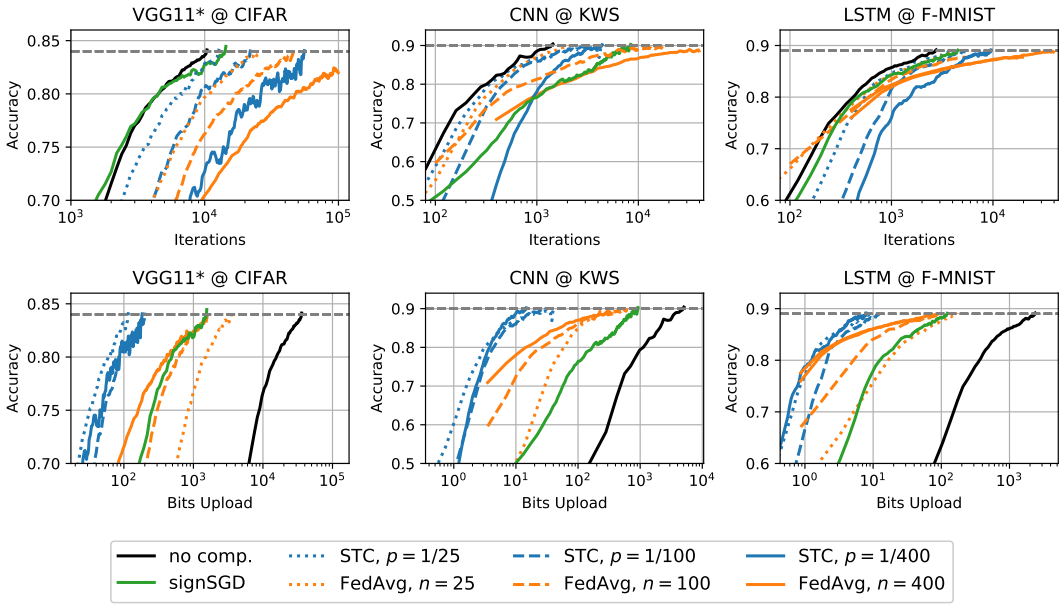


FIGURE 4.5: Convergence speed of Federated Learning with compressed communication in terms of training iterations (top) and uploaded bits (bottom) on three different benchmarks (left to right) in an iid Federated Learning environment with 100 clients and 10% participation fraction. For better readability the validation error curves are average-smoothed with a step-size of 5. On all benchmarks STC requires the least amount of bits to converge to the target accuracy.

trained on Fashion-MNIST. We can see that even if all clients hold iid data STC still manages to achieve the desired target accuracy within a smallest communication budget out of all methods. STC also converges faster in terms of training iterations than the versions of Federated Averaging with comparable compression rate. We see that both for Federated Averaging and STC we face a trade-off between the number of training iterations ("computation") and the number of communicated bits ("communication"). On all investigated benchmarks however STC is pareto-superior to Federated Averaging in the sense that for any fixed iteration complexity it achieves a lower (upload) communication complexity.

Table 4.2 shows the amount of upstream- and downstream-communication required to achieve the target accuracy for the different methods in megabytes. On the CIFAR learning task STC at a sparsity rate of $p = 0.0025$ only communicates 183.9 MB worth of data, which is a reduction in communication by a factor of $\times 199.5$ as compared to the baseline with requires 36696 MB and Federated Averaging ($n = 100$) which still requires 1606 MB. Federated Averaging with a delay period of 1000 steps does not achieve the target accuracy within the given iteration budget.

4.4 Summary & Limitations

Federated Learning for mobile and IoT applications is a challenging task, as generally little to no control can be exerted over the properties of the learning environment.

In this chapter we demonstrated, that the convergence behavior of current methods for communication-efficient Federated Learning is very sensitive to these properties. On a variety of different data sets and model architectures we observed that the convergence speed of Federated Averaging drastically decreases in learning environments where the clients either hold non-iid subsets of data, are forced to train

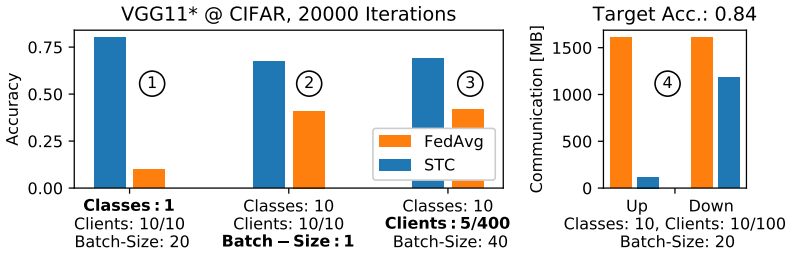


FIGURE 4.6: Summary of the advantageous properties of the STC algorithm. Left: Accuracy achieved by VGG11* on CIFAR after 20000 iterations of Federated Training with Federated Averaging and Sparse Ternary Compression for three different configurations of the learning environment. Right: Upstream and downstream communication necessary to achieve a validation accuracy of 84% with Federated Averaging and STC on the CIFAR benchmark under iid data and a moderate batch-size.

on small mini-batches or where only a small fraction of clients participates in every communication round. We demonstrated that the sparse k-nary compression protocol, introduced in the previous chapter, is far more robust to the above mentioned peculiarities of the learning environment than the very popular Federated Averaging method. Moreover, sparse k-nary compression converges faster than Federated Averaging both with respect to the amount of training iterations and the amount of communicated bits, even if the clients hold iid data and use moderate batch sizes during training.

Our approach can be understood as an alternative paradigm for communication-efficient federated optimization which relies on high-frequent low-volume instead of low-frequent high-volume communication. As such it is particularly well suited for Federated Learning environments which are characterized by low latency and low bandwidth channels between clients and server.

It must be noted however, that the downstream compression rate, achieved with sparse k-nary compression, scales inversely with the amount of communication rounds that are skipped by any client on average. Consequently our method can not provide significant reduction in downstream communication in situations where the client population is very large and only few clients participate in each round.

Furthermore, communication in sparse k-nary compression still fundamentally scales with the model size. Thus for very large-scale models, like transformer networks, communication overhead can still be prohibitive. In the next chapter, we will introduce an efficient Federated Learning algorithm, for which communication is independent of the model size, enabling Federated Learning of even larger models in highly communication-constrained applications.

Lessons Learned

- If the training data is distributed among the clients in a non-iid way, sparse communication protocols such as STC distinctively outperform Federated Averaging across all Federated Learning environments.
- The same holds true if clients are forced to train on small mini-batches (e.g. because of memory constraints). In these situations STC outperforms Federated Averaging even if the client's data is iid.
- STC achieves preferable performance over Federated Averaging if the client participation rate is low, as it converges more stable and quickly in both the iid and non-iid regime.
- On the other hand, a low client participation rate also leads to less efficient downstream compression for STC.
- STC is generally most useful in situations where the communication is bandwidth-constrained or costly (metered network, limited battery), as it does achieve a certain target accuracy within the minimum amount of communicated bits even on iid data (cf. Fig. 4.5, Tab. 4.2).
- Federated Averaging in return should be used if the communication is latency-constrained or if the client participation is expected to be very low.

Chapter 5

Communication-Efficient Federated Distillation

In the previous chapter, we learned that the communication bottleneck induced by frequently exchanging training information between the participating clients over limited bandwidth channels, is one of the most challenging obstacles in Federated Learning. The main reason for this is that the communication of both local gradients and model updates, which are the basic unit of information for gradient descent based distributed training methods like distributed SGD and Federated Averaging, is communication intensive and requires $\mathcal{O}(|\theta|)$ bits of information, where $|\theta|$ is the model size.

While different algorithmic approaches to tackle this problem have been proposed in the literature under the umbrella of efficient Federated Learning, the fundamental issue of scaling to larger models remains. As we have learned in Chapter 2, the recently proposed framework of Federated Distillation (Jeong et al., 2018; Li and Wang, 2019; Itahara et al., 2020; Lin et al., 2020b; Chen and Chao, 2020) has fundamentally different communication properties. As client knowledge in Federated Distillation is aggregated not by means of parameter averaging, but instead by distillation from the aggregated student predictions, communication will scale with the size of the distillation data set instead of the model size. This can be advantageous, especially if very large models are being trained. In this chapter, we will closely examine Federated Distillation with respect to its communication properties and introduce a set of improvements, which further reduce communication in both the upstream and the downstream, without negatively affecting the training performance.

This chapter is based on

- Felix Sattler, Arturo Marban, Roman Rischke, and Wojciech Samek (2021a). "CFD: Communication-Efficient Federated Distillation via Soft-Label Quantization and Delta Coding". In: *IEEE Transactions on Network Science and Engineering*, pp. 1–1. ISSN: 2327-4697. DOI: 10.1109/TNSE.2021.3081748. URL: https://dx.doi.org/10.1109/TNSE.2021.3081748

5.1 Federated Distillation Frameworks

Albeit their novelty, Federated Distillation techniques have been used in several existing works already. In the following we present a comprehensive overview on these existing techniques. Most relevant for the studied multi-round protocol for diverse models in this chapter is the protocol proposed by Itahara et al. (Itahara et al., 2020), which is based on ideas from Jeong et al. (Jeong et al., 2018) and mostly follows the steps described in section 2.3.3 with the sole exception that client models

are required to participate in every round and are not kept synchronized during local distillation by means of random seeds.

The similar protocol by Jeong et al. (Jeong et al., 2018) and Seo et al. (Seo et al., 2020) is instead based on locally accumulated logits per label, which are aggregated by the server. Furthermore, instead of exploiting these global logits for refining the local models by direct distillation, they are used for regularizing the local training in the next round. Similarly, Bistritz et al. (Bistritz, Mann, and Bambos, 2020) use distillation on an unlabelled public data set for regularizing on-device learning in a peer-to-peer network. Guha et al. (Guha, Talwalkar, and Smith, 2019) propose a one-shot distillation method for convex models, where the server distills the locally optimized client models in a single round based on an unlabelled data set.

The recently proposed FedMD by Li and Wang (Li and Wang, 2019) and Cronus by Chang et al. (Chang et al., 2019) also address knowledge distillation in Federated Learning through aggregated logits for a public data set. In FedMD, the clients train in each round first on the public data set and then on the private data set for personalization and communicate afterwards their model output on the public data set to the server, where the aggregation of the uploaded logits for the next round is performed. For the initial pretraining in FedMD the public data set is required to be labelled, whereas in the communication rounds after initialization the aggregated logits from the clients serve as soft-labels for the public data set. In Cronus, however, each client uses the local data set and the soft-labelled public data set jointly for local training.

Lin et al. (Lin et al., 2020b) apply ensemble distillation on top of Federated Averaging to refine the global server model resulting in fewer communication rounds compared to benchmark Federated Averaging methods. Although leveraging the power of ensemble distillation for robust model fusion and data augmentation, their method, called FedDF, is based on the classical Federated Averaging protocol with all the mentioned consequences w.r.t. the communication-efficiency.

Chen and Chao (Chen and Chao, 2020) introduce FedBE, where the server creates Bayesian model ensembles based on the uploaded client models, instead of directly averaging the client models as in FedAvg, and uses an unlabelled data set to distill one global student model from a Bayesian model ensemble created from the teacher models. This global model is transferred back to the clients as initialization for the next round of local training. Like FedDF, this approach however is more closely related to classical Federated Averaging than to Federated Distillation, from a communication perspective, since all clients have to train the same model architecture and the model parameters are communicated up- and downstream.

5.2 Investigating the Communication Properties of Federated Distillation

In this section we investigate the communication properties of Federated Distillation. In each round of Federated Distillation, clients will send a set of soft-label predictions

$$Y_i^{pub} = \{f_{\theta_i}(x) | x \in X^{pub}\} \tag{5.1}$$

to the server, and receive back aggregated soft-labels. The total amount of communication necessary to transfer this information in each round is given by the product of the distillation data set size and the average amount of bits required to store the

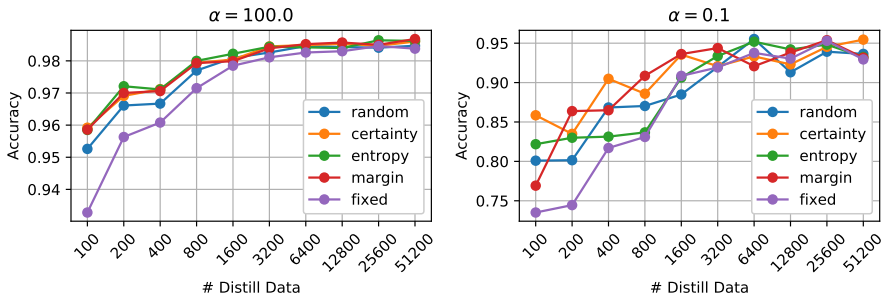


FIGURE 5.1: Effect of distillation data set size and different active selection strategies on the Federated Distillation performance.

value of one soft-label

$$\mathtt{b}_{total} = |X^{pub}| \times (H(Y_i) + \eta). \tag{5.2}$$

Hereby $|X^{pub}|$ is the size of the distillation data set, $H(Y_i)$ is the entropy of the soft-labels, and η indicates the coding inefficiency. In conventional Federated Distillation as proposed in (Jeong et al., 2018; Itahara et al., 2020), the soft-label information is stored at 32-bit floating-point precision, and thus, we have

$$\mathtt{b}_{total} = |X^{pub}| \times \dim(\mathcal{Y}) \times 32 \text{ bit}. \tag{5.3}$$

Following eq. (5.2), a reduction of the communication overhead can be achieved by either

- reducing the size of the distillation data set,
- reducing the entropy of the soft-labels, or
- improving the efficiency of the coding technique.

In this section, we will look at all three of these determining factors and investigate their relative impact on the Federated Learning performance.

In the preliminary experiments performed in this section, we consider Federated Learning settings with 20 clients among which we split the training data according to the Dirichlet strategy described in Chapter 2. More details on the experiment setup can be found in Section 5.4.

5.2.1 Distillation Data Set Size

As the communication overhead in Federated Distillation is directly proportional to the number of data points used for distillation, restricting the size of the distillation data is the most straight-forward way to reduce communication. It is commonly known however, that in machine learning (and deep learning in particular), the size of the training data set has strong impact on the generalization capacity of any trained classifier (Vapnik, 2013). The machine learning discipline of active learning has developed techniques to systematically select samples from a larger pool of data for training with the goal to achieve better performance with fewer samples of data. Here, we adapt four popular active learning techniques to the setting of Federated Distillation and compare their performance when used to select distillation data sets

of different sizes. Let

$$\text{top}_n[x \mapsto \Psi(x)] : \mathcal{D} \to \mathcal{D} \tag{5.4}$$

be the operator that maps a data set to one of its subsets of size n, by selecting the top n elements according to the criterion $x \mapsto \Psi(x)$. Then, we can define the "entropy", "certainty", and "margin" selection strategies as follows:

$$D_n^{entropy} = \text{top}_n[x \mapsto H(f_\theta(x))](X^{pub}) \tag{5.5}$$

$$D_n^{certainty} = \text{top}_n[x \mapsto -\max(f_\theta(x))](X^{pub}) \tag{5.6}$$

$$D_n^{margin} = \text{top}_n[x \mapsto \max_2(f_\theta(x)) - \max(f_\theta(x))](X^{pub}) \tag{5.7}$$

Hereby, $H(p) = -\sum_i p_i \log(p_i)$ denotes the entropy, $\max(p)$ represents the maximum value in the vector of probabilities p, and $\max_2(p) = \max(p \setminus \{\arg\max(p)\})$ denotes the second-largest element of p. For instance $D_n^{certainty}$ selects those n data points from X^{pub} for which the maximum likelihood prediction $\max(f_\theta(x))$ is assigned the lowest certainty. We also consider the selection strategy of picking n data-points at random in each round.

In each communication round of Federated Distillation, we select a subset of n data points for distillation, according to one of the above strategies based on the model θ, which was used in the previous round. The results of this experiment are shown in Figure 5.1. As we can see, the performance of Federated Distillation strongly depends on the size of the distillation data set. On the other hand, the effect of using active learning strategies to systematically select data points is rather low. While in the iid regime ($\alpha = 100.0$) the active learning strategies slightly improve the Federated Distillation performance, the situation is rather unclear in the non-iid regime ($\alpha = 0.1$). From these results, we conclude that in most situations, the performance gains obtained by using active learning strategies do not justify the additional computational overhead incurred by these techniques (i.e., evaluating $f_\theta(x)$ on the entire accessible distillation data). Thus, in the remainder of this chapter, we will restrict our analysis to randomly selected distillation data sets of fixed size.

5.2.2 Soft-Label Quantization

Quantization is a popular technique to reduce communication and has been successfully applied in Federated Averaging to reduce the size of the parameter updates (Konecný et al., 2016; Sattler et al., 2019; Xu et al., 2020). Quantization techniques, however, so far have not been applied to Federated Distillation. Here we consider constrained uniform quantization to reduce the entropy of the communicated soft-labels. Let $p \in \mathcal{Y}$ be a vector of soft-label probabilities. Then, we obtain the quantized soft-label q via constrained uniform quantization (Widrow, Kollar, and Liu, 1996) as follows

$$q = \mathcal{Q}_b(p) = \arg \min_{\substack{q_i \in \{\frac{l}{2^b-1}, l \in 0,..,2^b-1\} \\ \sum_i q_i = 1}} \|q - p\|_1 \tag{5.8}$$

The optimization problem above can be solved in log-linear time. In case the optimization problem in (5.8) does not have a unique solution, we randomly break the tie. As can be easily seen, for $b = 1$, the quantization operator $\mathcal{Q}_b$ is equivalent to

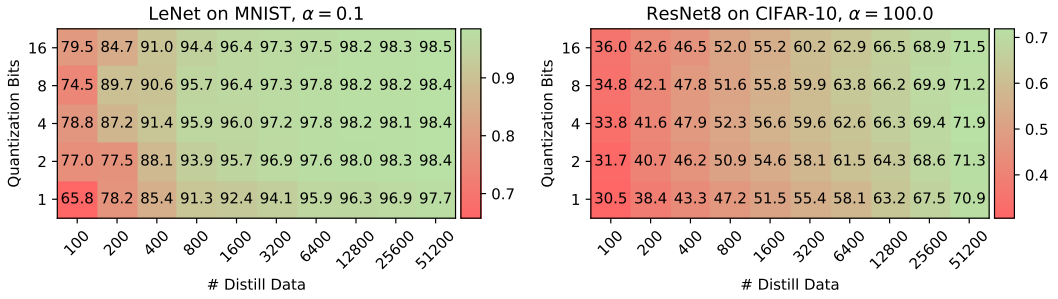


FIGURE 5.2: Effect of distillation data set size and quantization strength on training performance in Federated Distillation.

the maximum vote:

$$\mathcal{Q}_1(p)_i = \begin{cases} 1 & \text{if } i = \arg\max(p) \\ 0 & \text{else} \end{cases} \tag{5.9}$$

Constrained uniform quantization as defined above reduces the number of bits required to communicate any vector of probabilities from 32-bits $\times$ $\dim(\mathcal{Y})$ to b-bits $\times$ $\dim(\mathcal{Y})$.

Figure 5.2 shows the effect of different distillation data set sizes and quantization levels on the model accuracy after a fixed number of communication rounds. From this data, we notice two interesting trends. Firstly, we observe that, while reducing the number of quantization bits by half has the same effect on the communication overhead as reducing the size of the distillation data by the same amount, the former strategy has a much lower impact on the training performance. This result holds across different levels of quantization and distillation data set sizes. Second, as the size of the distillation data set increases, the harmful effects of quantization vanish. For instance, in the MNIST data set (Figure 5.2, left plot), the experimental findings suggest that for $n \geq 6400$ distillation data points, the model performance remains strong for any quantization level (with a small accuracy degradation at the highest compression levels). Similar effects can be observed when training ResNet-8 the CIFAR-10 data set (Figure 5.2, right plot), where in some cases higher compression rates even lead to slight improvements in accuracy. Moreover, notice that in both data sets, when the maximum number of distillation samples are available ($n = 51200$), the strongest compression level (i.e., 1-bit quantization) only incurs less then 1% accuracy degradation on both data sets.

These results indicate that as a means for reducing communication, quantization should be strictly preferred over distillation data set reduction, especially if one has access to a large distillation data set. In the following, we will concentrate our analysis on 1-bit quantization using the compression operator $\mathcal{Q}_1$.

5.2.3 Efficient Encoding

In this section we investigate efficient lossless coding techniques to minimize the size of the compressed soft-label representations. As shown in eq. (5.9), applying the compression operator $\mathcal{Q}_1$ to a vector of probabilities p results in a one-hot vector of size $\dim(\mathcal{Y})$. As this one-hot vector can also be represented by an integer number between 1 and $\dim(\mathcal{Y})$, a straight-forward encoding process would comprise of

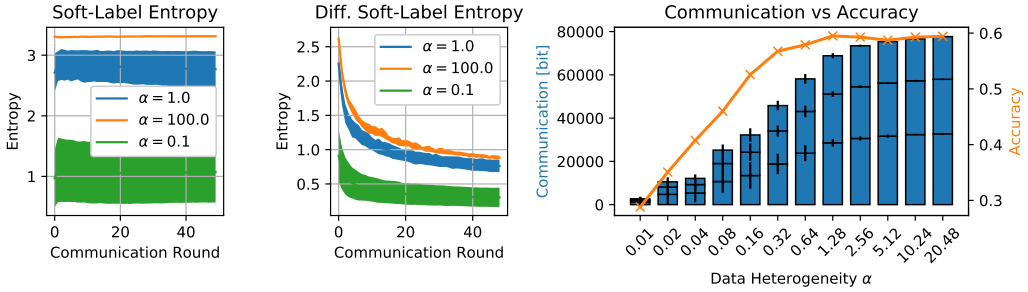


FIGURE 5.3: Panels 1 and 2: Evolution of the soft-label entropy when training ResNet-8 on the CIFAR-10 data set, at different levels of data-heterogeneity. When communicating compressed soft-labels directly (panel 1), the entropy stays constant over the course of training. In contrast, when using differential soft-labels (panel 2), the entropy steadily decreases. Panel 3: Upstream communication (vertical bars) and model accuracy (red curve), at different levels of data heterogeneity α, for ResNet-8 trained on CIFAR-10. Communication varies by more than an order of magnitude between the most homogeneous ($\alpha = 163.84$) and the most heterogeneous ($\alpha = 0.01$) setting.

communicating

$$\tilde{Y}_i = \{\mathcal{Q}_1(f_{\theta_i}(x)) | x \in X^{pub}\} \tag{5.10}$$

as an array of $|X^{pub}|$ integer values, using up $|X^{pub}| \times \log_2(\dim(\mathcal{Y}))$ bits of data in total.

This however is only an upper bound on the true entropy $H(\tilde{Y}_i)$, which highly depends on the distribution of max predictions in $\tilde{Y}_i$. Figure 5.3, panel 1, shows the development of $H(\tilde{Y}_i)$ over the course of 50 communication rounds for a Federated Learning problem with 20 clients training ResNet-8 on CIFAR-10 at different levels of data heterogeneity. As we can see, the true entropy is well below the theoretical maximum of $\log_2(\dim(\mathcal{Y})) = \log_2(10)$ and decreases with increasing heterogeneity α, down to around $H(\tilde{Y}_i) \approx 1$ at $\alpha = 0.1$. This behaviour is expected, as the labels in the client training data, and consequently also their predictions $\tilde{Y}_i$, get more concentrated with increasing heterogeneity in the data. Additional knowledge about the distribution of $\tilde{Y}_i$ can be used to further reduce the entropy. Since Federated Distillation is empirically known to converge (Lin et al., 2020b; Li and Wang, 2019), we can further expect there to be a growing overlap between the predictions made by a client in the current round T and those made in the previous round $T-1$.

High agreement between consecutive data points in a stream of data is a phenomenon commonly encountered in communication. The effect for instance can also be found in video data (Wiegand et al., 2003; Wiegand and Girod, 2001), where consecutive frames are often highly correlated. The canonical technique to exploit this pattern is differential coding (resp. delta coding or predictive coding) (Sayood, 2017), which relies on only communicating "new" information in order to achieve higher compression rates.

We apply loss-less delta coding to the quantized predictions of two consecutive rounds $\tilde{Y}^t$ and $\tilde{Y}^{t-1}$ by setting

$$(\hat{Y}^t)_l = \begin{cases} (\tilde{Y}^t)_l & \text{if } (\tilde{Y}^t)_l \neq (\tilde{Y}^{t-1})_l \\ 0 & \text{else} \end{cases} \quad \forall l \tag{5.11}$$

TABLE 5.1: Effect of the client model initialization on the maximum accuracy achieved in Federated Distillation after 50 communication rounds. Displayed are the mean and standard deviation of the accuracy computed from 3 experiments, with a client participation rate of 20%. Our proposed dual-distillation approach closes the gap between random model initialization and the (infeasible) initialization from the previous model state.

	α	Init Previous	Init Random	Dual Distill
MNIST	100.0	98.1%±0.0	96.7%±0.1	97.8%±0.0
	1.0	97.9%±0.1	96.4%±0.1	97.5%±0.1
	0.1	92.6%±1.4	90.1%±1.5	92.7%±1.2
CIFAR10	100.0	70.2%±0.4	68.5%±0.1	74.9%±0.2
	1.0	68.2%±0.6	66.0%±0.6	72.4%±0.7
	0.1	51.1%±3.5	48.2%±2.3	56.1%±3.0

and measuring the entropy (in slight abuse of notation this assumes an arbitrary but fixed ordering of the set $\tilde{Y}$ and the same distillation data set X^{pub} to be used in all rounds). It should be noted, that all of the information contained in $\tilde{Y}^t$ can be retained from $\hat{Y}^t$ by comparing with the previous message $\hat{Y}^{t-1}$. This only requires minor additional bookkeeping by the central server, which is typically assumed to have access to strong computational resources.

Figure 5.3, panel 2, shows the development of the entropy of the differential updates $H(\hat{Y}_i)$. As we can see, the differential soft-label entropy behaves exactly as predicted and $H(\hat{Y}_i)$ is lower than $H(\tilde{Y}_i)$ from the first round on and smoothly decreases over the course of training. We note that, curiously, the development of the differential soft-label entropy over time can be very accurately predicted via the functional relation $H(\hat{Y}^t) \approx ct^{-d}$ for some constants c, d. We were able to replicate this behaviour across different model architectures and Federated Learning settings, hinting at an interesting underlying mathematical relationship, which could be the subject of future studies.

Figure 5.3, panel 3, explores in more detail the influence of data heterogeneity on the amount of communication. It displays the upstream communication in the first three rounds of Federated Distillation with quantization and differential soft-label encoding. The resulting model accuracy is also given (indicated by the red curve). As we can see, the amount of communication monotonically decreases when lowering the value of α (thus increasing the data heterogeneity), with more than an order of magnitude difference between the most homogeneous ("iid") setting at $\alpha = 163.84$ and the most heterogeneous setting at $\alpha = 0.01$.

5.2.4 Efficient Downstream Communication

So far we have only considered the upstream communication from the clients to the server. While in most Federated Learning settings with mobile and IoT devices, the uplink channel is more constrained than the downlink channel, it is still desirable to reduce the downstream communication as much as possible.

As we have learned in the previous chapter, compressing the down-link in Federated Learning however is challenging, as clients may run out of sync if they are not initialized with the same model in every round (the importance of a common model initialization has been famously demonstrated in (McMahan et al., 2017)). If the participation rate is below 100% and clients do not participate in every round, state

information (like the model state θ) becomes stale. To keep the client models synchronized under these conditions, clients need to either download the latest master-model θ from the server in every round (resulting in high downstream communication) or alternatively randomly re-initialize their local models in every round using a common random seed (resulting in performance degradation).

To illustrate this point, Table 5.1 shows the maximum accuracy achieved after 50 communication rounds of Federated Distillation with three different client initialization schemes and three different neural networks at varying levels of data heterogeneity. As we can see, initializing the client models randomly before distillation ("Init Random") achieves worse performance than using the distilled model from the previous round ("Init Prev.") as the initialization point. To close the performance gap, we propose a novel dual distillation technique, which avoids de-synchronization of client models at arbitrary participation rates.

Let I_t be the set of clients participating in one particular communication round t and

$$Y^{pub} = \frac{1}{|I_t|} \sum_{i \in I_t} \tilde{Y}_i^{pub} \tag{5.12}$$

be the corresponding set of aggregated soft-labels. In dual distillation, instead of directly sending the aggregated soft-labels to the clients, the server first performs a distillation step of it's own

$$\theta_S^t \leftarrow \text{train}(\theta_S^{t-1}, X^{pub}, Y^{pub}) \tag{5.13}$$

using the model θ_S^{t-1}, which was distilled in the previous round, as initialization point. This way the training information stored in θ_S^{t-1} is not lost. Then, the server computes soft-labels using the newly distilled model,

$$Y_S^{pub} = \{f_{\theta_S}(x) | x \in X^{pub}\} \tag{5.14}$$

and sends them to the clients. Starting from a random initialization, the participating clients then distill from the server predictions to mimic the server model

$$\theta \leftarrow \text{train}(\theta_0, X^{pub}, Y_S^{pub}) \tag{5.15}$$

This way the clients are indirectly initialized with all the accumulated training information stored in θ_S, before going into the next round of local training.

This allows us now to communicate soft-labels in upstream and downstream and appreciate the resulting communication savings in both directions, without loss of accuracy (cf. Table 5.1). To further reduce the amount of downstream communication, we can also quantize the server soft-labels Y_S^{pub} before communication, using the same constrained compression operator $\mathcal{Q}_{b_{down}}$ that we used in the upstream. We emphasize that dual distillation is only necessary if client participation is below 100%. Furthermore in Federated Learning the server is typically assumed to have much stronger computational resources than the clients, thus the workload of training an additional server model can mostly be neglected.

Figure 5.4 shows the effects of different levels of upstream and downstream quantization on the training performance of LeNet trained on MNIST, using Federated Distillation after 20 communication rounds. As we can see in the iid setting with $\alpha = 100.0$, downstream quantization appears to have a slightly stronger effect on the model performance than upstream quantization, with a maximum accuracy drop of

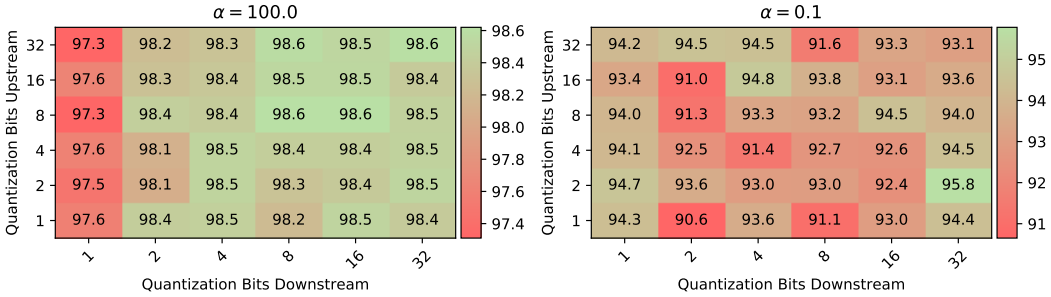


FIGURE 5.4: Effect of different levels of upstream and downstream quantization on the performance of CFD. LeNet trained on MNIST, at homogeneous ($\alpha = 100.0$) and heterogeneous ($\alpha = 0.1$) distributions of data. Displayed is the maximum accuracy achieved after 20 communication rounds.

1% at the highest quantization level, i.e., $b_{down} = 1$. In contrast, in the non-iid setting with $\alpha = 0.1$, it is more difficult to observe such a trend. Here, the strongest levels of upstream and downstream compression outperform the uncompressed FD. Thus, it appears that using quantization in both directions, upstream and downstream, is a promising technique for reducing communication.

5.3 Compressed Federated Distillation

In this section, we combine the insights of the previous section and propose Compressed Federated Distillation (CFD). CFD extends the conventional Federated Distillation framework by the following five techniques:

1. **Distill data curation (Alg. 8 - 1)**: We select a fixed random subset X^{pub} of the available distillation data for training. This subset is not varied over the course of training.

2. **Upstream quantization (Alg. 8 - 14)**: We reduce the bit-width of the client soft-labels by applying the constrained quantization operator $\mathcal{Q}$ (eq. (5.8)).

3. **Delta coding (Alg. 8 - 14)**: The quantized soft-labels are encoded using an efficient arithmetic entropy coding technique, like CABAC (Marpe, Schwarz, and Wiegand, 2003). Additionally, we use delta coding (eq. (5.11)), to further reduce the entropy of the quantized soft-label information $\tilde{Y}_i$.

4. **Dual distillation (Alg. 8 - 19, 20)**: In every round, we distill a server model θ_S from the aggregated soft-labels. This server model accumulates training information from all previous communication rounds. The clients are then trained to match the predictions of this server model. This avoids loss of information in settings where clients do not participate in every round.

5. **Downstream quantization (Alg. 8 - 21)**: We apply constrained quantization $\mathcal{Q}$ also to the predictions of the server model before sending them down to the clients. The clients then, starting from a random initialization, are trained to mimic the predictions of the server model.

The training procedure is illustrated in Figure 5.5 and formally described in Algorithm 8.

The performance of our algorithm in every round t is determined from the distilled model θ_S on the validation data set.

Algorithm 8 Compressed Federated Distillation

init: Set upstream and downstream precision b_{up} and b_{down}. Every client, C_i, holds a different local data set, $D_i = (X_i, Y_i)$, as well as the common public data set, X^{pub}, with size $|X^{pub}| = n$.

for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., M\}$ **in parallel do**
    Client C_i does:
    $\theta \leftarrow$ random_init() # Initialize
    if $t > 1$ **then**
      download$_{S \rightarrow C_i}(\tilde{Y}_S^{pub})$
      $\theta \leftarrow \text{train}(\theta, X^{pub}, \tilde{Y}_S^{pub})$ # Distillation
    end if
    $\theta_i \leftarrow \text{train}(\theta, X_i, Y_i)$ # Local Training
    $Y_i^{pub} \leftarrow f_{\theta_i}(X^{pub})$ # Compute Soft-Labels
    $\tilde{Y}_i^{pub} \leftarrow \mathcal{Q}_{b_{up}}(Y_i^{pub})$ # Compress Soft-Labels
    upload$_{C_i \rightarrow S}(\tilde{Y}_i^{pub})$ # Upload
  end for
  Server S does:
  $Y^{pub} \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} \tilde{Y}_i^{pub}$ # Aggregate
  $\theta_S \leftarrow \text{train}(\theta_S, X^{pub}, Y^{pub})$ # Server Distillation
  $Y_S^{pub} \leftarrow f_{\theta_S}(X^{pub})$ # Compute Soft-Labels
  $\tilde{Y}_S^{pub} \leftarrow \mathcal{Q}_{b_{down}}(Y_S^{pub})$ # Compress Soft-Labels
end for
return θ_S

5.4 Experiments

In this section we empirically evaluate our proposed Compressed Federated Distillation method and compare its performance against the natural baselines of Federated Averaging (McMahan et al., 2017) and Federated Distillation (Itahara et al., 2020). The experimental setup is given as follows:

Data sets and models: We evaluate CFD on both federated image and text classification problems with large scale convolutional and transformer neural networks, respectively. For our image classification problems we experiment with the following combinations of client- and/ distillation data: (MNIST / EMNIST (Cohen et al., 2017)) and (CIFAR-10 / STL-10 (Coates, Ng, and Lee, 2011)). In both cases the distribution of the distillation data deviates from the one of the client data, as it would in realistic Federated Learning scenarios (MNIST contains handwritten digits, EMNIST contains handwritten characters, CIFAR-10 and STL-10 both contain different types of natural images). For our text classification problems we use disjoint splits of the SST2 (Socher et al., 2013) and AG-News (Zhang, Zhao, and LeCun, 2015a) data sets for client training, distillation, and validation, respectively. We train LeNet- (LeCun et al., 1989), VGG-type (Simonyan and Zisserman, 2015), AlexNet-type (Krizhevsky, Sutskever, and Hinton, 2012) and ResNet-type (He et al., 2016) architectures with and without batch-normalization layers. The Alexnet, ResNet-18 and VGG-16 models used in our experiments contain 23.2M, 11.1M, and 15.2M parameters respectively. For our text classification experiments we fine-tune DistilBERT (Sanh et al., 2019), a popular transformer model with $\sim$66M parameters.

Federated Learning environment and data partitioning: For image classification problems, we consider Federated Learning settings with 20 clients. In all experiments,

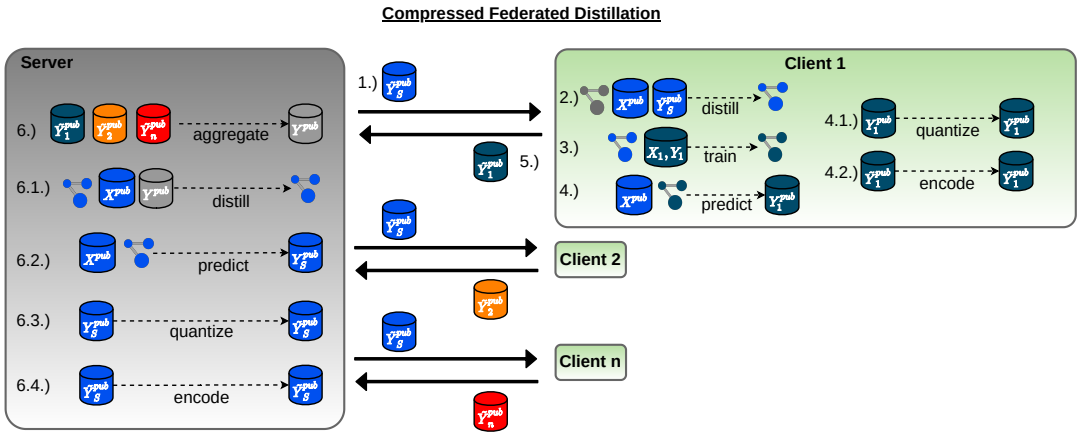


FIGURE 5.5: Illustration of our proposed Compressed Federated Distillation method. CFD employs distill data curation (see Sec. 5.2.1), soft-label quantization (4.2., see Sec 5.2.2) and delta-coding (4.3., see Sec. 5.2.3) to minimize the communication from the clients to the server. Furthermore, CFD uses dual distillation (6.1., 6.2., see Sec. 5.2.4) to keep clients synchronized in situations when full client participation in every round can not be ensured. On top of that CFD also uses quantization (6.3., see Sec. 5.2.4) and delta-coding (6.4.) in the downstream, to reduce the communication from the server to the clients.

we split the training data evenly among the clients according to the Dirichlet strategy described in Chapter 2, which allows us to smoothly adapt the level of non-iid-ness in the client data using the Dirichlet parameter α. We experiment with values for α varying between 100.0 and 0.01. A value of $\alpha = 100.0$ results in almost identical label distributions, while setting $\alpha = 0.01$ results in a split, where the vast majority of data on every client stems from one single class (see Figure 2.5 for an illustration). For image classifiers, we vary the clients' participation rate (in every round) between 40% and 100%, and train for 50 communication rounds. For language models, we set the number of clients to 10 and the participation rate to 100%, and train for a total of 10 communication rounds. As a standard convention in FL, the validation data follow the clients' training data distribution (and not the distillation data distribution).

Optimization details: For the sake of simplicity, in all image classification tasks, we use the popular Adam (Kingma and Ba, 2015) optimizer with a fixed learning rate of 0.001 across all baselines and for both, the distillation and training on local private data. While a dedicated selection of optimizer and optimization hyperparameters might improve performance, our goal here is to give a fair comparison between the different Federated Learning algorithms. For language models, we perform one epoch of distillation with Adam, using a learning rate of 1×10^{-5} and no weight decay. The clients' models are trained over one epoch with SGD in both scenarios, Federated Distillation and Federated Averaging, by setting the learning rate and momentum to 0.001 and 0.9, respectively.

Baselines: We compare the performance of our method, Compressed Federated Distillation (CFD), with respect to the two natural baselines: Federated Averaging (FA) (McMahan et al., 2017) and Federated Distillation (FD) (Itahara et al., 2020). For CFD, we test two configurations: For CFD-1-32 we only quantize the upstream communication by setting $b_{up} = 1$ and $b_{down} = 32$. For CFD-1-1 we quantize both the upstream and downstream communication and set $b_{up} = 1$ and $b_{down} = 1$. We also investigate the effects of using delta coding (as described in section 5.2.3). CFD methods that use delta coding are indicated by CFD_{Δ}.

TABLE 5.2: Upstream and downstream communication in [MB], necessary to achieve accuracy targets in Federated Learning on the CIFAR-10 data set, across different neural network models and levels of data heterogeneity ($\alpha = 100.0$, 1.0, and 0.1). The Federated Learning setting consists of 20 clients and a participation rate of 40%. For the distillation based methods, 80000 data points from the STL-10 data set are used as distillation data. A value of "n.a." signifies that the method did not achieve the target accuracy within 50 communication rounds. The number of communication rounds, necessary to achieve the target accuracy is given in parenthesis. Minimum communication per training scenario is highlighted in bold face.

Model/ Data set	Target Acc.	α	Up/ Down	FA	FD	CFD-1-32	CFD$_\Delta$-1-32	CFD-1-1	CFD$_\Delta$-1-1
				Communication [MB]					
ResNet-18 CIFAR-10	0.71	100.0	up	760.3 (17)	44.8 (14)	0.56 (17)	**0.40** (17)	1.36 (41)	0.82 (41)
			down	760.3 (17)	44.8 (14)	54.4 (17)	54.4 (17)	1.36 (41)	**0.39** (41)
	0.68	1.0	up	1028.7 (23)	48.0 (15)	0.37 (13)	**0.28** (13)	0.64 (22)	0.43 (22)
			down	1028.7 (23)	48.0 (15)	41.6 (13)	41.6 (13)	0.72 (22)	**0.34** (22)
	0.45	0.1	up	1520.7 (34)	16.0 (5)	0.09 (7)	**0.08** (7)	0.52 (41)	0.40 (41)
			down	1520.7 (34)	16.0 (5)	22.4 (7)	22.4 (7)	0.99 (41)	**0.92** (41)
VGG-16 CIFAR-10	0.8	100.0	up	671.1 (11)	32.0 (10)	0.40 (12)	**0.29** (12)	0.76 (23)	0.47 (23)
			down	671.1 (11)	32.0 (10)	38.4 (12)	38.4 (12)	0.76 (23)	**0.24** (23)
	0.78	1.0	up	1281.3 (21)	28.8 (9)	0.38 (13)	**0.28** (13)	0.56 (19)	0.37 (19)
			down	1281.3 (21)	28.8 (9)	41.6 (13)	41.6 (13)	0.62 (19)	**0.27** (19)
	0.48	0.1	up	2928.6 (48)	25.6 (8)	0.11 (9)	**0.09** (9)	0.43 (34)	0.35 (34)
			down	2928.6 (48)	25.6 (8)	28.8 (9)	28.8 (9)	0.77 (34)	**0.75** (34)
AlexNet CIFAR-10	0.68	100.0	up	n.a.	89.6 (28)	0.94 (29)	**0.74** (29)	n.a.	n.a.
			down	n.a.	**89.6** (28)	92.80 (29)	92.8 (29)	n.a.	n.a.
	0.64	1.0	up	n.a.	38.4 (12)	0.61 (21)	**0.49** (21)	0.76 (26)	0.62 (26)
			down	n.a.	38.4 (12)	67.2 (21)	67.2 (21)	0.84 (26)	**0.42** (26)
	0.44	0.1	up	n.a.	6.40 (2)	0.09 (6)	**0.08** (6)	0.11 (7)	0.10 (7)
			down	n.a.	6.40 (2)	19.20 (6)	19.20 (6)	0.17 (7)	**0.15** (7)
DistillBert SST	0.88	100.0	up	267.8 (1)	0.269 (1)	**0.004** (1)	0.006 (1)	0.029 (7)	0.044 (7)
			down	267.8 (1)	0.269 (1)	0.269 (1)	0.26 (1)	**0.029** (7)	0.044 (7)
	0.88	1.0	up	803.4 (3)	0.53 (2)	**0.008** (2)	0.012 (2)	0.038 (10)	0.057 (10)
			down	803.4 (3)	0.539 (2)	0.539 (2)	0.539 (2)	**0.042** (10)	0.063 (10)
DistillBert AG-News	0.91	100.0	up	535.64 (2)	1.92 (2)	**0.030** (2)	0.035 (2)	**0.030** (2)	0.035 (2)
			down	535.64 (2)	1.92 (2)	1.92 (2)	1.92 (2)	**0.030** (2)	0.035 (2)
	0.91	1.0	up	1071.28 (4)	4.80 (5)	0.142 (10)	0.168 (10)	**0.101** (7)	0.119 (7)
			down	1071.28 (4)	4.80 (5)	9.60 (10)	9.60 (10)	**0.105** (7)	0.121 (7)

Evaluation Metrics: As is custom in communication-efficient Federated Learning literature (Konecný et al., 2016; Caldas et al., 2018a; Sattler et al., 2020b), we report cumulative communication of the different FL methods. Given this general metric, other quantities of interest like wall-clock time or energy consumption can be approximated for any given hardware setup and/ or communication infrastructure. For the Baseline FD and our methods, CFD and CFD$_\Delta$, we measure only the communication of soft-labels Y^{pub} and explicitly ignore the communication cost of transferring the unlabeled public data set X^{pub} to the participating clients. While clients technically need to download this data once before training, it is not subject to the same constraints as the Federated Learning process. In communication-sensitive applications, X^{pub} could already be stored on the devices long before the federated training process starts, and thus, the timing of its communication is much less critical. Other work (Lin et al., 2020b) also demonstrates that X^{pub} can be automatically generated on the

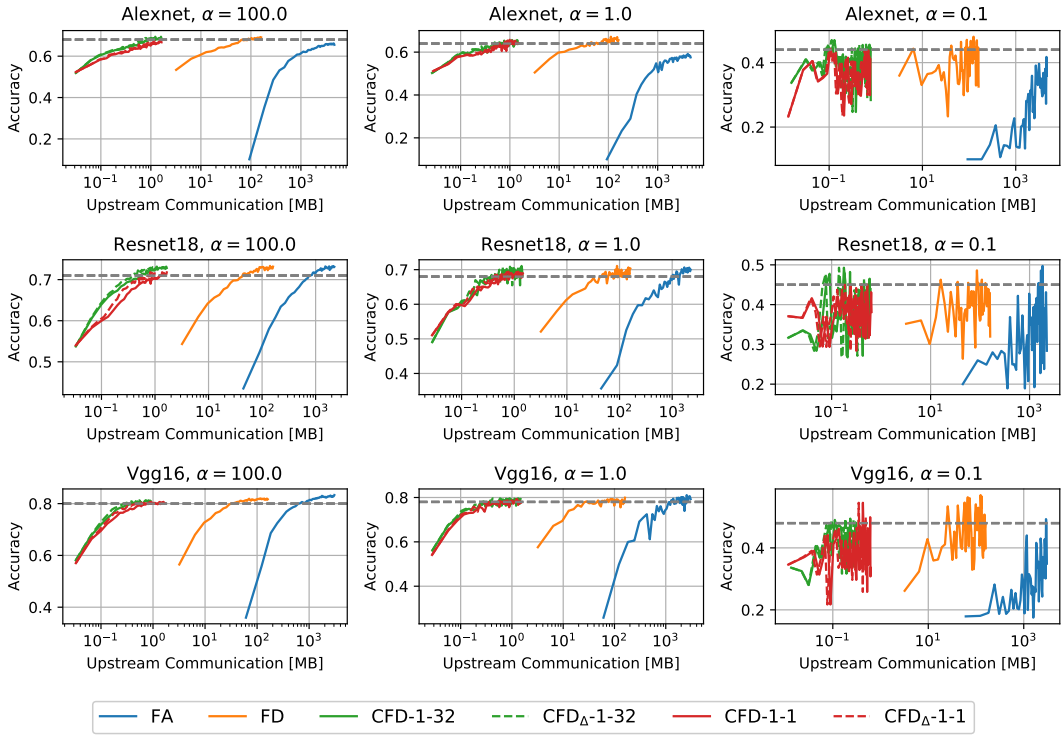


FIGURE 5.6: Model performance as a function of communicated bits for our proposed CFD method and baselines methods FA and FD in Federated Learning on the CIFAR-10 data set, across different neural network models and levels of data heterogeneity. The Federated Learning setting consists of 20 clients with a 40% participation rate. For the distillation based methods, 80000 data points from the STL-10 data set were used.

clients using Generative Adversarial Networks.

5.4.1 Image Classification Results

We first investigate the communication properties of CFD on image classification benchmarks. Table 5.2 shows the amount of upstream and downstream bits, as well as the number of communication rounds, required to achieve fixed accuracy targets for Alexnet, ResNet-18, and VGG-16 on CIFAR-10, at different levels of data heterogeneity between the clients. The corresponding training curves are given in Figure 5.6. As we can see, CFD is drastically more communication-efficient than the baselines FA and FD in all tested scenarios. For instance, for VGG-16 and $\alpha = 100.0$, CFD$_\Delta$-1-1 achieves a target accuracy of 80% by cumulatively communicating only 0.47 MB on average, from the clients to the server, and only 0.24 MB on average, from the server to the clients. This is particularly remarkable, as one single transfer of the parameters of VGG-16 already takes up 61.01 MB. To achieve the same 80% accuracy target, FA requires 671.16 MB of cumulative communication in both the upstream and the downstream, translating to more than three orders of magnitude in communication savings for CFD. When directly comparing with FD, which requires 32.00 MB, CFD still reduces the communication by about two orders of magnitude. Similar results can be observed for the two other tested neural networks, ResNet-18 and Alexnet. On Alexnet, FA even underperforms CFD w.r.t. to the maximum achieved accuracy and misses the accuracy target of 68%.

The communication savings are even larger in the non-iid settings with $\alpha = 0.1$, where FA is known to perform poorly (Sattler et al., 2020b). For instance, when training ResNet-18 at $\alpha = 0.1$, FA requires 1520.70 MB to achieve the accuracy target of 45%. In contrast, CFD$_\Delta$-1-32 requires only 0.08 MB to achieve the same accuracy, corresponding to a reduction in communication by a factor of $\times$19943.

In all investigated settings, CFD$_\Delta$ methods that use delta coding are more efficient than those that do not. For instance, for VGG-16 and $\alpha = 1.0$, delta coding can bring down the cumulative upstream-communication required to achieve 78% accuracy from 0.38 MB to 0.28 MB for CFD-1-32. On the same benchmark, delta coding also reduces the cumulative upstream-communication from 0.56 MB to 0.37 MB for CFD-1-1.

As can be seen in Figure 5.6, the heavily compressed CFD can keep up with the uncompressed baselines FD and FA w.r.t. maximum achieved accuracy on most benchmarks.

5.4.2 Language Model Results

We fine-tune, DistilBERT, a popular large-scale transformer model, on the SST2 and AG-News data sets. In these experiments, we consider a Federated Learning setting with 10 clients, 100% participation rate, and total of 10 communication rounds. Table 5.2 shows the upstream and downstream communication cost (in MB) necessary to achieve certain accuracy targets across different levels of data heterogeneity, as well as the required number of communication rounds. As we can see, similar as on the image classification problems, CFD requires several orders of magnitude less communication than FA and FD in both the upstream and downstream to achieve fixed performance targets. For instance, when compared to FA, CFD achieves communication savings of up to $\times$66955 on the SST2 data set and $\times$17855 on the AG-News data set, with negligible accuracy degradation. In some situations, this comes at the cost of an increased number of total communication rounds. We also notice that in this particular set of experiments, delta-coding (see CFD$_\Delta$-1-1 and CFD$_\Delta$-1-32), slightly increases the communication overhead when compared to regular CFD. This effect is probably caused by the small number of classes in the data sets (i.e., SST2 contains 2 classes, while AG-News contains 4 classes), which limits the benefits of delta-coding.

5.5 Summary & Limitations

In this chapter we explored the communication properties of Federated Distillation and demonstrated that drastic compression gains can be achieved via soft-label quantization and delta coding. For instance, on language modeling tasks, we demonstrated that our proposed Compressed Federated Distillation method can reduce the cumulative communication necessary to achieve fixed performance targets from 1071.28 MB to 0.101 MB when compared to the very popular Federated Averaging algorithm. This corresponds to a reduction in communication by over four orders of magnitude. Similar compression rates were obtained in our investigated image classification problems on popular convolutional neural networks.

It is important to note however, that the favorable communication properties of all Federated Distillation methods, come at the cost of additional computational overhead caused by the local distillation. This additional computational overhead might be prohibitive in Federated Learning environments where clients have limited

computational resources, or where the number of clients is high and/or the number of data points per client is low. Furthermore, it needs to be re-emphasized that Federated Distillation additionally requires the server and the clients to have access to an unlabeled set of distillation data. Transferring this data to the clients will typically incur additional communication overhead (although it can be done before training when devices are idle). Related approaches (Lin et al., 2020b), which demonstrate that the distillation data can be automatically generated on the clients using Generative Adversarial Networks should be further investigated in this context.

Due to these reasons it needs to be carefully considered for every application, which of the two paradigms - Federated Averaging or Federated Distillation - is more suitable for the problem at hand.

In the next chapter we will venture away from the challenges and intricacies of communication efficient training and take a broader look at Federated Learning, challenging some of it's core assumptions and in doing so, open up new ways to treat data heterogeneity.

Lessons Learned

- Communication in Federated Distillation scales proportional to the size of the distillation data set and is independent of the model size, which can be advantageous if large neural network models are trained.
- Reducing the size of the distillation data set beyond a certain threshold has significant negative impact on the training performance.
- Soft-label information communicated in Federated Distillation can be heavily compressed by means of constrained quantization, at only a marginal loss of training performance.
- The differential soft-label entropy monotonically decreases as the jointly trained model converges toward a stationary solution. This can be exploited using delta-coding techniques to further reduce communication.

Chapter 6

Clustered Federated Learning

Federated Learning not only needs to be efficient, it should also provide maximum utility to the participants in order to incentivise participation. This however can be challenging to achieve under the constraints of the conventional Federated Learning framework, where only one single model is trained from the combined data of all participants. In many situations it may instead be beneficial to train different models for different sub-groups of the client population, which can optimally adapt to structured variations in the local data of each individual. In this chapter we will challenge the single-model assumption typically made in Federated Learning, and instead consider these more general Federated Learning settings, where client data exhibits a clustering structure. By treating Federated Learning as a multi-task learning problem we will open up new possibilities for model personalization and improve robustness of the training process to faulty and malicious clients.

This chapter is based on

- Felix Sattler, Klaus-Robert Müller, and Wojciech Samek (2020). "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13. DOI: 10.1109/TNNLS.2020.3015958. URL: https://doi.org/10.1109/TNNLS.2020.3015958
- Felix Sattler, Klaus-Robert Müller, Thomas Wiegand, and Wojciech Samek (2020a). "On the Byzantine Robustness of Clustered Federated Learning". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8861–8865. DOI: 10.1109/ICASSP40776.2020.9054676. URL: http://dx.doi.org/10.1109/ICASSP40776.2020.9054676 ©2020 IEEE

6.1 Generalizing the Federated Learning Assumption

Federated Learning generally assumes that a single model can be trained from the combined data of all participants of a distributed training task. Given a model $f_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by $\theta \in \Theta$ and a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ we can formally state the conventional Federated Learning assumption as follows:

Assumption 1. ***("Federated Learning"):*** *There exists a parameter configuration $\theta^* \in \Theta$, that (locally) minimizes the risk on all clients' data generating distributions at the same time:*

$$R_i(\theta^*) \leq \min_{\substack{\theta \\ \|\theta - \theta^*\| < \varepsilon}} R_i(\theta), \quad i = 1, .., M \tag{6.1}$$

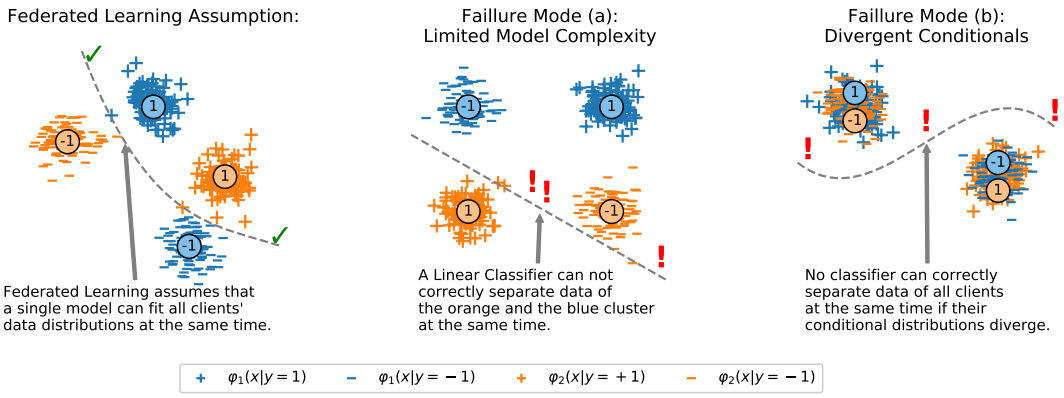


FIGURE 6.1: Two toy cases in which the Federated Learning Assumption is violated. Blue points belong to clients which follow φ_1 while orange points belong to clients which follow φ_2. Left: Federated Learning assumes that a single model can fit all clients' data distributions at the same time. Middle: Federated XOR-problem. An insufficiently complex model is not capable of fitting all clients' data distributions at the same time. Right: If different clients' conditional distributions diverge, no single model can fit all distributions at the same time. In all cases however the data of clients belonging to the same cluster can be easily separated.

for some $\varepsilon > 0$. *Hereby*

$$R_i(\theta) = \int l(f_\theta(x), y) d\varphi_i(x, y) \tag{6.2}$$

is the risk function associated with distribution φ_i.

It is easy to see that this assumption is not always satisfied. Concretely it is violated if either (a) the model f_θ is not expressive enough to fit all distributions at the same time or (b) clients have disagreeing conditional distributions $\varphi_i(y|x) \neq \varphi_j(y|x)$. Simple counter examples for both cases are presented in Figure 6.1.

In the following we will call a set of clients and their data generating distributions φ *congruent* (with respect to f and l) if they satisfy Assumption 1 and *incongruent* if they don't.

We argue that Assumption 1 is frequently violated in real Federated Learning applications, especially given the fact that in Federated Learning clients (a) can hold arbitrary non-iid data, which can not be audited by the centralized server due to privacy constraints and (b) typically run on limited hardware which puts restrictions on the model complexity. For illustration consider the following practical scenarios:

Varying Preferences: Assume a scenario where every client holds a local data set of images of human faces and the goal is to train an 'attractiveness' classifier on the joint data of all clients. Naturally, different clients will have varying opinions about the attractiveness of certain individuals, which corresponds to disagreeing conditional distributions on all clients' data. Assume for instance that one half of the client population thinks that people wearing glasses are attractive, while the other half thinks that those people are unattractive. In this situation one single model will *never* be able to accurately predict attractiveness of glasses-wearing people for all clients at the same time (cf. also Figure 6.1 right).

Limited Model Complexity: Assume a number of clients are trying to jointly train a language model for next-word prediction on private text messages. In this scenario the statistics of a client's text messages will likely vary a lot based on demographic factors, interests, etc. For instance, text messages composed by teenagers will typically

exhibit different statistics than those composed by elderly people. An insufficiently expressive model will not be able to fit the data of all clients at the same time (cf. also Figure 6.1 middle).

Presence of Adversaries: A special case of incongruence is given, if a subset of the client population behaves in an adversarial manner. In this scenario the adversaries could deliberately alter their local data distribution in order to encode arbitrary behavior into the jointly trained model, thus affecting the model decisions on all other clients and causing potential harm (Sattler et al., 2020a).

The goal in *Federated Multi-Task Learning* is to provide every client with a model that optimally fits it's local data distribution. In all of the above described situations the ordinary Federated Learning framework, in which all clients are treated equally and only one single global model is learned, is not capable of achieving this goal.

In order to incorporate the above presented problems with incongruent data generating distributions, we suggest to generalize the conventional Federated Learning Assumption:

Assumption 2. ***("Clustered Federated Learning"):*** *There exists a partitioning* $\mathcal{C} = \{c_1, .., c_K\}$, $\dot{\bigcup}_{k=1}^{K} c_k = \{1, .., M\}$ *of the client population, such that every subset of clients* $c \in \mathcal{C}$ *satisfies the conventional Federated Learning Assumption.*

In this chapter we present Clustered Federated Learning, a novel algorithmic framework that is able to deal with Federated Multi-Task learning problems that satisfy Assumtion 2. By identifying the hidden clustering structure, Clustered Federated Learning allows clients with similar data to profit from one another, while minimizing the harmful interference between clients with dissimilar data.

We additionally refer the reader to Table 6.1, which gives a detailed comparison between our proposed method and related methods for Federated Multi-Task learning.

6.2 Clustering based on Gradient Signals

In this section, we address the question of how to solve distributed learning problems that satisfy Assumption 2 (which generalizes the Federated Learning Assumption 1). This will require us to identify the correct partitioning $\mathcal{C}$, which at first glance seems like a daunting task, as under the Federated Learning paradigm the server doesn't have access to the clients' data, their data generating distributions or any meta information thereof.

6.2.1 Cosine Similarity based Bi-Partitioning

An easier task than trying to directly infer the entire clustering structure $\mathcal{C}$, is to find a correct bi-partitioning in the sense of the following definition:

Definition 2. Let $M \geq K \geq 2$ and

$$I : \{1, .., M\} \rightarrow \{1, .., K\}, i \mapsto I(i) \tag{6.3}$$

be the mapping that assigns a client i to it's data generating distribution $\varphi_{I(i)}$. Then we call a bi-partitioning $c_1 \dot{\cup} c_2 = \{1, .., M\}$ with $c_1 \neq \emptyset$ and $c_2 \neq \emptyset$ *correct* if and only if

$$I(i) \neq I(j) \quad \forall i \in c_1, j \in c_2. \tag{6.4}$$

In other words, we call a bi-partitioning *correct*, if clients with the same data generating distribution end up in the same cluster. It is easy to see, that the clustering $\mathcal{C} = \{c_1, .., c_k\}$ can be obtained after exactly $K - 1$ correct bi-partitions. In the following we will demonstrate that there exists an explicit criterion based on which a correct bi-partitioning can be inferred. To see this, let us first look at the following simplified Federated Learning setting with M clients, in which the data on every client was sampled from one of two data generating distributions φ_1, φ_2 such that

$$D_i \sim \varphi_{I(i)}(x, y). \tag{6.5}$$

Every client is associated with an empirical risk function

$$r_i(\theta) = \sum_{(x,y)\in D_i} l_\theta(f(x), y) \tag{6.6}$$

which approximates the true risk arbitrarily well if the number of data points on every client is sufficiently large

$$r_i(\theta) \approx R_{I(i)}(\theta). \tag{6.7}$$

For demonstration purposes let us first assume equality in (6.7). Then the Federated Learning objective becomes

$$F(\theta) := \sum_{i=1}^{M} \frac{|D_i|}{|D|} r_i(\theta) = a_1 R_1(\theta) + a_2 R_2(\theta) \tag{6.8}$$

with $a_1 = \sum_{i,I(i)=1} |D_i|/|D|$ and $a_2 = \sum_{i,I(i)=2} |D_i|/|D|$ and $D = \bigcup_{i=1,..,M} D_i$. Under standard assumptions it has been shown (Li et al., 2020b; Sahu et al., 2018) that the Federated Learning optimization protocol described in Algorithm 2 converges to a stationary point θ^* of the Federated Learning objective. In this point it holds that

$$0 = \nabla F(\theta^*) = a_1 \nabla R_1(\theta^*) + a_2 \nabla R_2(\theta^*) \tag{6.9}$$

Now we are in one of two situations. Either it holds that $\nabla R_1(\theta^*) = \nabla R_2(\theta^*) = 0$, in which case we have simultaneously minimized the risk of all clients. This means φ_1 and φ_2 are congruent and we have solved the distributed learning problem. Or, otherwise, it has to hold

$$\nabla R_1(\theta^*) = -\frac{a_2}{a_1} \nabla R_2(\theta^*) \neq 0 \tag{6.10}$$

and φ_1 and φ_2 are incongruent. In this situation the *cosine similarity* between the gradient updates of any two clients is given by

$$\begin{aligned} \alpha_{i,j} := \alpha(\nabla r_i(\theta^*), \nabla r_j(\theta^*)) &:= \frac{\langle \nabla r_i(\theta^*), \nabla r_j(\theta^*) \rangle}{\|\nabla r_i(\theta^*)\| \|\nabla r_j(\theta^*)\|} \\ &= \frac{\langle \nabla R_{I(i)}(\theta^*), \nabla R_{I(j)}(\theta^*) \rangle}{\|\nabla R_{I(i)}(\theta^*)\| \|\nabla R_{I(j)}(\theta^*)\|} \\ &= \begin{cases} 1 & \text{if } I(i) = I(j) \\ -1 & \text{if } I(i) \neq I(j) \end{cases} \end{aligned} \tag{6.11}$$

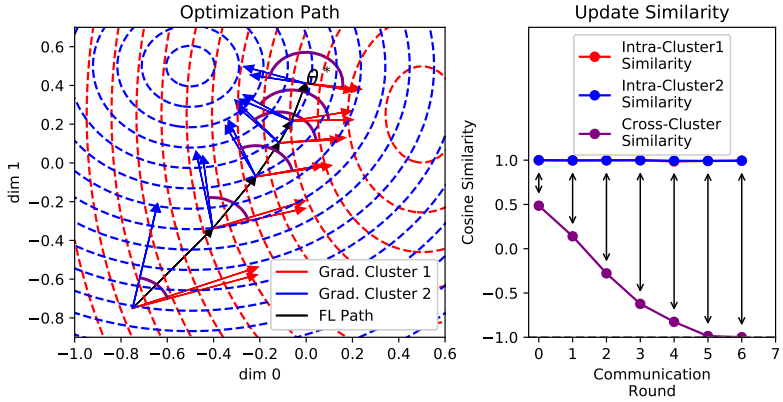


FIGURE 6.2: Optimization path of Federated Learning with four clients, belonging to two different clusters with incongruent data distributions and cosine similarity between their respective gradient updates. Federated Learning converges to a stationary solution of the FL objective θ^* where the gradients of the two clients are of positive norm and point into opposite directions (6.11). While the cosine similarity between gradient updates from the same cluster stays more or less constant throughout the federated training process, the cosine similarity between the gradient updates from different clusters quickly decreases.

Consequently, a correct bi-partitioning is given by

$$c_1 = \{i | \alpha_{i,0} = 1\},\ c_2 = \{i | \alpha_{i,0} = -1\}. \tag{6.12}$$

This consideration provides us with the insight that, in a stationary solution of the Federated Learning objective θ^*, we can *distinguish clients based on their hidden data generating distribution by inspecting the cosine similarity between their gradient updates*. For a visual illustration of the result we refer to Figure 6.2.

If we drop the equality assumption in (6.7) and allow for an arbitrary number of data generating distributions K, we obtain the following generalized version of result (6.11):

Theorem 3 (Separation Theorem). *Let $D_1, .., D_M$ be the local training data of M different clients, each data set sampled from one of K different data generating distributions $\varphi_1, .., \varphi_K$, such that $D_i \sim \varphi_{I(i)}(x, y)$. Let the empirical risk on every client approximate the true risk at every stationary solution of the Federated Learning objective θ^* s.t.*

$$\|\nabla R_{I(i)}(\theta^*)\| > \|\nabla R_{I(i)}(\theta^*) - \nabla r_i(\theta^*)\| \tag{6.13}$$

and define

$$\gamma_i := \frac{\|\nabla R_{I(i)}(\theta^*) - \nabla r_i(\theta^*)\|}{\|\nabla R_{I(i)}(\theta^*)\|} \in [0, 1) \tag{6.14}$$

Then there exists a bi-partitioning $c_1^* \dot{\cup} c_2^* = \{1, .., M\}$ of the client population such that the maximum similarity between the updates from any two clients from different clusters can be

bounded from above according to

$$\begin{aligned}\alpha_{cross}^{max} &:= \min_{c_1 \dot\cup c_2 = \{1,..,M\}} \max_{i \in c_1, j \in c_2} \alpha(\nabla r_i(\theta^*), \nabla r_j(\theta^*)) \\ &= \max_{i \in c_1^*, j \in c_2^*} \alpha(\nabla r_i(\theta^*), \nabla r_j(\theta^*)) \\ &\leq \begin{cases} \cos(\frac{\pi}{K-1}) H_{i,j} + \sin(\frac{\pi}{K-1}) \sqrt{1 - H_{i,j}^2} & \text{if } H \geq \cos(\frac{\pi}{K-1}) \\ 1 & \text{else} \end{cases}\end{aligned} \tag{6.15}$$

with

$$H_{i,j} = -\gamma_i \gamma_j + \sqrt{1-\gamma_i^2}\sqrt{1-\gamma_j^2} \in (-1, 1]. \tag{6.16}$$

At the same time the similarity between updates from clients which share the same data generating distribution can be bounded from below by

$$\alpha_{intra}^{min} := \min_{\substack{i,j \\ I(i)=I(j)}} \alpha(\nabla_\theta r_i(\theta^*), \nabla_\theta r_j(\theta^*)) \geq \min_{\substack{i,j \\ I(i)=I(j)}} H_{i,j}. \tag{6.17}$$

The proof of Theorem 3 can be found in Appendix B.1.

Remark 1. *In the case with two data generating distributions ($K = 2$) the result simplifies to*

$$\alpha_{cross}^{max} = \max_{i \in c_1^*, j \in c_2^*} \alpha(\nabla_\theta r_i(\theta^*), \nabla_\theta r_j(\theta^*)) \leq \max_{i \in c_1^*, j \in c_2^*} -H_{i,j} \tag{6.18}$$

for a certain partitioning, respective

$$\alpha_{intra}^{min} = \min_{\substack{i,j \\ I(i)=I(j)}} \alpha(\nabla_\theta r_i(\theta^*), \nabla_\theta r_j(\theta^*)) \geq \min_{\substack{i,j \\ I(i)=I(j)}} H_{i,j} \tag{6.19}$$

for two clients from the same cluster. If additionally $\gamma_i = 0$ for all $i = 1, .., M$ then $H_{i,j} = 1$ and we retain result (6.11).

From Theorem 3 we can directly deduce a correct separation rule:

Corollary 2. *If in Theorem 3 K and γ_i, $i = 1, .., M$ are in such a way that*

$$\alpha_{intra}^{min} > \alpha_{cross}^{max} \tag{6.20}$$

then the partitioning

$$c_1, c_2 \leftarrow \arg\min_{c_1 \dot\cup c_2 = c} \left(\max_{i \in c_1, j \in c_2} \alpha_{i,j} \right). \tag{6.21}$$

is always *correct in the sense of Definition 2.*

Proof. Set

$$c_1, c_2 \leftarrow \arg\min_{c_1 \dot\cup c_2 = c} \left(\max_{i \in c_1, j \in c_2} \alpha_{i,j} \right) \tag{6.22}$$

and let $i \in c_1$, $j \in c_2$ then

$$\alpha_{i,j} \leq \alpha_{cross}^{max} < \alpha_{intra}^{min} = \min_{\substack{i,j \\ I(i)=I(j)}} \alpha_{i,j} \tag{6.23}$$

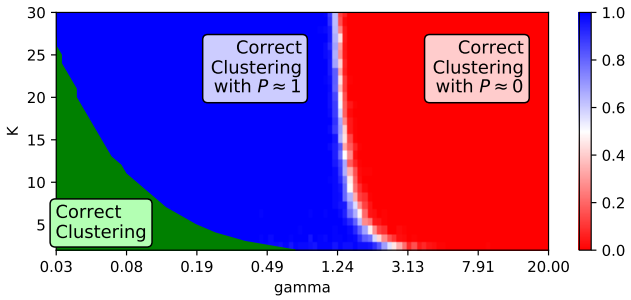


FIGURE 6.3: Clustering quality as a function of the number of data generating distributions K and the relative approximation noise γ. For all values of K and γ in the green area, CFL will *always* correctly separate the clients (by Theorem 3). For all values of K and γ in the blue area we find empirically that CFL will correctly separate the clients with probability close to 1.

and hence i and j can not have the same data generating distribution. □

This consideration leads us to the notion of the separation gap:

Definition 3 (Separation Gap)**.** Given a cosine-similarity matrix α and a mapping from client to data generating distribution I we define the notion of a separation gap

$$g(\alpha) := \alpha_{intra}^{min} - \alpha_{cross}^{max} \tag{6.24}$$

$$= \min_{\substack{i,j \\ I(i)=I(j)}} \alpha_{i,j} - \min_{c_1 \dot\cup c_2 = c} \left(\max_{i \in c_1, j \in c_2} \alpha_{i,j} \right) \tag{6.25}$$

Remark 2. *By Corollary 2 the bi-partitioning (6.21) will be correct in the sense of Definition 2 if the separation gap $g(\alpha)$ is greater than zero.*

Theorem 3 gives an estimate for the similarities in the *absolute worst-case*. In practice α_{intra}^{min} typically will be much larger and α_{cross}^{max} typically will be much smaller, especially if the parameter dimension $d := \dim(\Theta)$ is large. For instance, if we set $d = 10^2$ (which is still many orders of magnitude smaller than typical modern neural networks), $M = 3K$, and assume $\nabla R_{I(i)}(\theta^*)$ and $\nabla R_{I(i)}(\theta^*) - \nabla r_i(\theta^*)$ to be normally distributed for all $i = 1, .., M$ then experimentally we find (Figure 6.3) that

$$P[\text{"Correct Clustering"}] \geq P[g(\alpha) > 0] \approx 1 \tag{6.26}$$

even for large values of $K > 10$ and

$$\gamma_{max} := \max_{i=1,..,M} \gamma_i > 1. \tag{6.27}$$

This suggests that using the cosine similarity criterion (6.21) we can readily find a correct bi-partitioning c_1, c_2 even if the number of data generating distributions is high and the empirical risk on every client's data is only a very loose approximation of the true risk.

6.2.2 Distinguishing Congruent and Incongruent Clients

In order to appropriately generalize the classical Federated Learning setting, we need to make sure to only split up clients with *incongruent* data distributions. In the classical congruent non-iid Federated Learning setting described in (McMahan

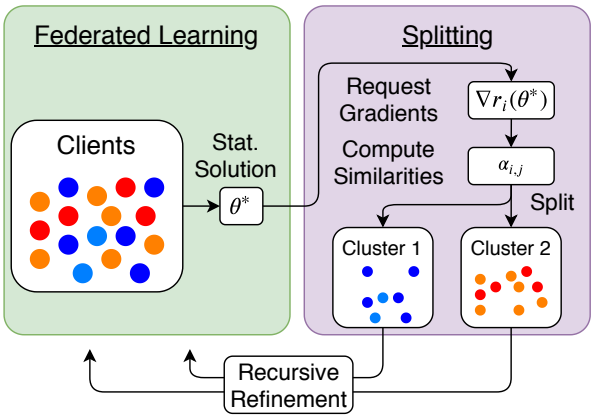


FIGURE 6.4: Schematic overview over the CFL Algorithm. By recursively bi-partitioning the client population into sub-groups of maximum dissimilarity, CFL produces a hierarchy of models of increasing specificity.

et al., 2017) where one single model can be learned, performance will typically degrade if clients with varying distributions are separated into different clusters due to the restricted knowledge transfer between clients in different clusters. Luckily we have a criterion at hand to distinguish the two cases. To realize this we have to inspect the gradients computed by the clients at a stationary point θ^*. When client distributions are incongruent, the stationary solution of the Federated Learning objective by definition can not be stationary for the individual clients. Hence the norm of the clients' gradients has to be strictly greater than zero. If conversely the client distributions are congruent, federated optimization will be able to jointly optimize all clients' local risk functions and hence the norm of the clients' gradients will tend towards zero as we are approaching the stationary point. Based on this observation we can formulate the following criteria which allow us to make the decision whether to split or not: Splitting should only take place if it holds that both (a) we are close to a stationary point of the FL objective

$$0 \leq \| \sum_{i=1,..,M} \frac{D_i}{|D|} \nabla_\theta r_i(\theta^*) \| < \varepsilon_1 \tag{6.28}$$

and (b) the individual clients are far from a stationary point of their local empirical risk

$$\max_{i=1,..,M} \| \nabla_\theta r_i(\theta^*) \| > \varepsilon_2 > 0 \tag{6.29}$$

We will also experimentally verify the clustering criteria (6.28) and (6.29) and give recommendations for the selection of the hyperparameters ε_1 and ε_2 in section 6.5.

In practice we have another viable option to distinguish the congruent from the incongruent case. As splitting is only performed after Federated Learning has converged to a stationary point θ^*, the conventional Federated Learning solution is always computed as part of Clustered Federated Learning. This means that if after splitting up the clients a degradation in model performance is observed, it is always possible to fall back to the Federated Learning solution. In this sense Clustered Federated Learning will always improve the Federated Learning performance (or perform equally well at worst).

6.2.3 Algorithm

Clustered Federated Learning recursively bi-partitions the client population in a top-down way: Starting from an initial set of clients $c = \{1, .., M\}$ and a parameter initialization θ_0, CFL performs Federated Learning according to Algorithm 10, in order to obtain a stationary solution θ^* of the FL objective. After Federated Learning has converged, the stopping criterion

$$0 \leq \max_{i \in c} \|\nabla_\theta r_i(\theta^*)\| < \varepsilon_2 \tag{6.30}$$

is evaluated. If criterion (6.30) is satisfied, we know that all clients are sufficiently close to a stationary solution of their local risk and consequently CFL terminates, returning the FL solution θ^*. If on the other hand, criterion (6.30) is violated, this means that the clients are incongruent and the server computes the pairwise cosine similarities α between the clients' latest transmitted updates according to equation (6.11). Next, the server separates the clients into two clusters in such a way that the maximum similarity between clients from different clusters is minimized

$$c_1, c_2 \leftarrow \arg\min_{c_1 \dot\cup c_2 = c} \big(\max_{i \in c_1, j \in c_2} \alpha_{i,j} \big). \tag{6.31}$$

This optimal bi-partitioning problem at the core of CFL can be solved in $\mathcal{O}(M^3)$ using Algorithm 9. Since in Federated Learning it is assumed that the server has far greater computational power than the clients, the overhead of clustering will typically be negligible.

As derived in section 6.2.1, a correct bi-partitioning can always be ensured if it holds that

$$\alpha_{intra}^{min} > \alpha_{cross}^{max}.$$

While the optimal cross-cluster similarity α_{cross}^{max} can be easily computed in practice, computation of the intra cluster similarity requires knowledge of the clustering structure and hence α_{intra}^{min} can only be estimated using Theorem 3 according to

$$\alpha_{intra}^{min} \geq \min_{\substack{i,j \\ I(i)=I(j)}} -\gamma_i\gamma_j + \sqrt{1-\gamma_i^2}\sqrt{1-\gamma_j^2} \tag{6.32}$$

$$\geq 1 - 2\gamma_{max}^2. \tag{6.33}$$

Consequently we know that the bi-partitioning will be correct if

$$\gamma_{max} < \sqrt{\frac{1-\alpha_{cross}^{max}}{2}}. \tag{6.34}$$

independent of the number of data generating distributions K. This criterion allows us to reject bi-partitionings, based on our assumptions on the approximation noise γ_{max} (which is an interpretable hyperparameter).

If criterion (6.34) is satisfied, CFL is recursively re-applied to each of the two separate groups starting from the stationary solution θ^*. Splitting recursively continues on until (after at most $K-1$ recursions) none of the sub-clusters violate the stopping criterion anymore, at which point all groups of mutually congruent clients $\mathcal{C} = \{c_1, .., c_K\}$ have been identified, and the Clustered Federated Learning problem characterized by Assumption 2 is solved. The communication burden of CFL thus increases at most

linearly with the number of splits and only after the Federated Learning solution is reached. The entire recursive procedure is presented in Algorithm 11. A schematic illustration is given in Figure 6.4.

6.3 Related Work

Federated Learning (McMahan et al., 2017; Konecný et al., 2016; Caldas et al., 2018b; Li, Wen, and He, 2019; Yang et al., 2019; Bonawitz et al., 2019) is currently the dominant framework for distributed training of machine learning models under communication- and privacy constraints. Federated Learning assumes the clients to be congruent, i.e. that one central model can fit all client's distributions at the same time. Different authors have investigated the convergence properties of Federated Learning in congruent iid and non-iid scenarios: (Lin et al., 2020a; Sattler et al., 2019; Sattler et al., 2020b) and (Zhao et al., 2018) perform an empirical investigation, (Li et al., 2020b), (Jiang and Agrawal, 2018), (Yu, Yang, and Zhu, 2019) and (Sahu et al., 2018) prove convergence guarantees. As outlined above conventional Federated Learning is not able to deal with the challenges of incongruent data distributions.

Algorithm 9 Optimal Bi-partition

input: Similarity Matrix $\alpha \in [-1,1]^{M,M}$
outout: bi-partitioning c_1, c_2 satisfying (6.21)

$s \leftarrow \text{argsort}(-\alpha[:]) \in \mathbb{N}^{M^2}$
$\mathcal{C} \leftarrow \{\{i\} | i = 1, .., M\}$
for $i = 1, .., M^2$ **do**
  $i_1 \leftarrow s_i \text{ div } M$
  $i_2 \leftarrow s_i \bmod M$
  $c_{tmp} \leftarrow \{\}$
  for $c \in \mathcal{C}$ **do**
    if $i_1 \in c$ **or** $i_2 \in c$ **then**
      $c_{tmp} \leftarrow c_{tmp} \cup c$
      $\mathcal{C} \leftarrow \mathcal{C} \setminus c$
    end if
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_{tmp}\}$
  if $|\mathcal{C}| = 2$ **then**
    return $\mathcal{C}$
  end if
end for

Algorithm 10 Federated Learning

input: initial parameters θ, set of clients c, $\varepsilon_1 > 0$
output: stationary solution θ^*

repeat
  for $i \in c$ **in parallel do**
    $\theta_i \leftarrow \theta$
    $\Delta\theta_i \leftarrow \text{SGD}(\theta_i, D_i) - \theta_i$
  end for
  $\theta \leftarrow \theta + \sum_{i \in c} \frac{|D_i|}{|D_c|} \Delta\theta_i$
until $\| \sum_{i \in c} \frac{|D_i|}{|D_c|} \Delta\theta_i \| < \varepsilon_1$
return θ^*

Algorithm 11 Clustered Federated Learning

input: initial parameters θ, set of clients c, $\gamma_{max} \in [0,1]$, $\varepsilon_2 > 0$
output: stationary solutions θ_i^* and cluster assignments c

$\theta^* \leftarrow \text{FederatedLearning}(\theta, c)$
$\alpha_{i,j} \leftarrow \frac{\langle \nabla r_i(\theta^*), \nabla r_j(\theta^*) \rangle}{\|\nabla r_i(\theta^*)\| \|\nabla r_j(\theta^*)\|}, i, j \in c$
$c_1, c_2 \leftarrow \arg\min_{c_1 \dot\cup c_2 = c} (\max_{i \in c_1, j \in c_2} \alpha_{i,j})$
$\alpha_{cross}^{max} \leftarrow \max_{i \in c_1, j \in c_2} \alpha_{i,j}$
if $\max_{i \in c} \|\nabla r_i(\theta^*)\| \geq \varepsilon_2$ **and** $\sqrt{\frac{1-\alpha_{cross}^{max}}{2}} > \gamma_{max}$ **then**
  $\theta_i^*, i \in c_1 \leftarrow \text{ClusteredFederatedLearning}(\theta^*, c_1)$
  $\theta_i^*, i \in c_2 \leftarrow \text{ClusteredFederatedLearning}(\theta^*, c_2)$
else
  $\theta_i^* \leftarrow \theta^*, i \in c$
end if
return $\theta_i^*, i \in c$

TABLE 6.1: Qualitative comparison between methods for Federated Multi-Task Learning.

Desideratum	(Smith et al., 2017b)	(Ghosh et al., 2019)	**CFL**
Works with arbitrary *non-convex* objective functions.	✗	✗	✓
Does not require changes to the FedAvg protocol to be made.	✗	✓	✓
No major computational overhead for the clients.	-	✓	✓
Cluster number need not be known a priori.	✓	✗	✓
Theoretical guarantees on the clustering quality.	-	✓	✓
Can be implemented with formal privacy guarantees.	?	?	✓
Only performs clustering if necessary.	✗	✗	✓
Can handle varying client populations.	?	?	✓

✓ = satisfies property, ✗ = does not satisfy property, **?** = not investigated, **-** = does not apply

Other distributed training frameworks (Koloskova et al., 2020; Stich, Cordonnier, and Jaggi, 2018; Smith et al., 2017a) are facing the same issues.

The natural framework for dealing with incongruent data is Multi-Task Learning (Caruana, 1997; Jacob, Vert, and Bach, 2009; Kumar and III, 2012). An overview over recent techniques for multi-task learning in deep neural networks can be found in (Ruder, 2017). However all of these techniques are applied in a centralized setting in which all data resides at one location and the server has full control over and knowledge about the optimization process. Smith et al. (Smith et al., 2017b) present MOCHA, which extends the multi-task learning approach to the Federated Learning setting, by explicitly modeling client similarity via a correlation matrix. However their method relies on alternating bi-convex optimization and is thus only applicable to convex objective functions and limited in it's ability to scale to massive client populations. Corinzia et al. (Corinzia and Buhmann, 2019) model the connectivity structure between clients and server as a Bayesian network and perform variational inference during learning. Although their method can handle non-convex models, it is expensive to generalize to large federated networks as the client models are refined sequentially.

Finally, Ghosh et al. (Ghosh et al., 2019) propose a clustering approach, similar to the one presented in this chapter. However their method differs from ours in the following key aspects: Most significantly they use $l2$-distance instead of cosine similarity to determine the distribution similarity of the clients. This approach has the severe limitation that it only works if the client's risk functions are convex and the minima of different clusters are well separated. Their method also is not able to distinguish congruent from incongruent settings. This means that the method will incorrectly split up clients in the conventional congruent non-iid setting described in (McMahan et al., 2017). Furthermore, their approach is not adaptive in the sense that the decision whether to cluster or not is made already after the first communication round. In contrast, our method can be applied to arbitrary Federated Learning problems with non-convex objective functions. We also note that we have provided theoretical considerations that allow a systematic understanding of the novel CFL

framework. For a detailed qualitative comparison between Federated Multi-Task learning methods, we refer to Table 6.1.

6.4 Implementation Considerations

In this section we consider practical implementation details of our method. Concretely we will demonstrate that CFL can be implemented without making modifications to the Federated Learning communication protocol and without compromising the privacy of the participating clients. We will also demonstrate that our method is flexible enough to handle client populations that vary over time.

6.4.1 Weight-Updates as generalized Gradients

Theorem 3 makes a statement about the cosine similarity between *gradients* of the empirical risk function. In Federated Learning however, due to constraints on the communication budged of the client devices, instead commonly weight-updates

$$\Delta\theta = \mathrm{SGD}(\theta_0, D) - \theta_0 \tag{6.35}$$

obtained from performing multiple epochs of local training, are computed and communicated (McMahan et al., 2017). In order to deviate as little as possible from the classical Federated Learning algorithm it would hence be desirable to generalize result 3 to weight-updates. It is commonly conjectured (see e.g. (Lin et al., 2018)) that accumulated mini-batch gradients approximate the full-batch gradient of the objective function. Indeed, for a sufficiently smooth loss function and low learning rate, a weight update computed over one epoch approximates the direction of the true gradient by Taylor expansion: Given a split

$$\bigcup_{\tau=0,..,n_b-1} D_\tau = D \tag{6.36}$$

of a clients' data D into n_b disjoint batches we have after one epoch of SGD

$$\begin{aligned}\Delta\theta &= \mathrm{SGD}(\theta_0, D) - \theta_0 \\ &= -\sum_{\tau=0}^{n_b-1} \eta \nabla_\theta r(\theta_\tau, D_\tau) \\ &\approx -\sum_{\tau=0}^{n_b-1} \eta \nabla_\theta r(\theta_0, D_\tau) = -\eta \nabla_\theta r(\theta_0, D).\end{aligned} \tag{6.37}$$

with

$$\theta_\tau = \theta_{\tau-1} - \eta \nabla_\theta r(\theta_{\tau-1}, D_{\tau-1}), \quad \tau > 0. \tag{6.38}$$

In the remainder of this work we will compute cosine similarities between weight-updates instead of gradients according to

$$\alpha_{i,j} := \frac{\langle \Delta\theta_i, \Delta\theta_j \rangle}{\|\Delta\theta_i\| \|\Delta\theta_j\|}, i, j \in c \tag{6.39}$$

Our experiments in section 6.5 will demonstrate that computing cosine similarities based on weight-updates in practice surprisingly achieves *even better* separations than computing cosine similarities based on gradients.

6.4.2 Preserving Privacy

Every machine learning model carries information about the data it has been trained on. For example the bias term in the last layer of a neural network will typically carry information about the label distribution of the training data. Different authors have demonstrated that information about a client's input data ("x") can be inferred from the weight-updates it sends to the server via model inversion attacks (Bhowmick et al., 2018; Hitaj, Ateniese, and Perez-Cruz, 2017; Fredrikson, Jha, and Ristenpart, 2015; Carlini et al., 2018; Melis et al., 2019). In privacy sensitive situations it might be necessary to prevent this type of information leakage from clients to server with mechanisms like the ones presented in (Bonawitz et al., 2017). Luckily, Clustered Federated Learning can be easily augmented with an encryption mechanism that achieves this end. As both the cosine similarity between two clients' weight-updates and the norms of these updates are invariant to orthonormal transformations P (such as permutation of the indices),

$$\frac{\langle \Delta\theta_i, \Delta\theta_j \rangle}{\|\Delta\theta_i\| \|\Delta\theta_j\|} = \frac{\langle P\Delta\theta_i, P\Delta\theta_j \rangle}{\|P\Delta\theta_i\| \|P\Delta\theta_j\|} \tag{6.40}$$

a simple remedy is for all clients to apply such a transformation operator to their updates before communicating them to the server. After the server has averaged the updates from all clients and broadcasted the average back to the clients they simply apply the inverse operation

$$\frac{1}{|c|} \sum_{i \in c} \Delta\theta_i = P^{-1} \left(\frac{1}{|c|} \sum_{i \in c} P\Delta\theta_i \right) \tag{6.41}$$

and the Federated Learning protocol can resume unchanged. Other multi-task learning approaches require direct access to the client's data and hence can not be used together with encryption, which means that CFL has a distinct advantage in privacy sensitive situations.

6.4.3 Varying Client Populations and Parameter Trees

Up until now we always made the assumption that all clients participate from the beginning of training. Clustered Federated Learning however is flexible enough to handle client populations that vary over time.

In order to incorporate this functionality, the server, while running CFL, needs to build a parameter tree $T = (V, E)$ with the following properties:

- The tree contains a node $v \in V$ for every (intermediate) cluster c_v computed by CFL
- Both c_v and the corresponding stationary solution θ_v^* obtained by running the Federated Learning Algorithm 10 on cluster c_v are cached at node v
- At the root of the tree v_{root} resides the Federated Learning solution over the entire client population with $c_{v_{root}} = \{1, .., M\}$.

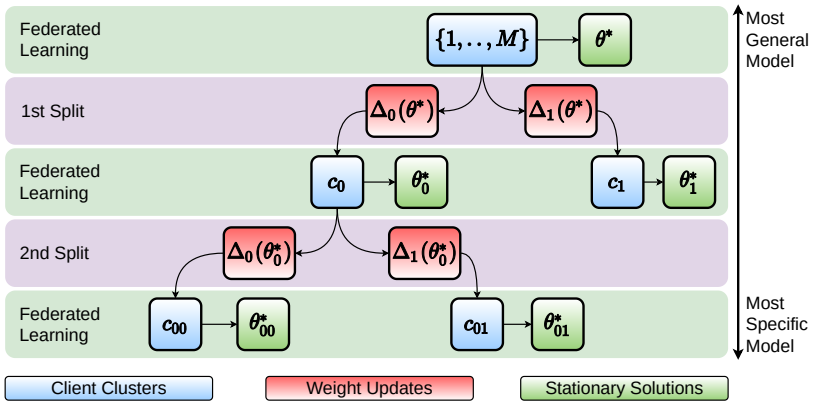


FIGURE 6.5: Example of a parameter tree created by Clustered Federated Learning. At the root node resides the conventional Federated Learning model, obtained by converging to a stationary point θ^* of the FL objective over all clients $\{1, .., M\}$. In the next layer, the client population has been split up into two groups, according to their cosine similarities and every subgroup has again converged to a stationary point θ_0^* respective θ_1^*. Branching continues recursively until no stationary solution satisfies the splitting criteria. In order to quickly assign new clients to a leaf model, at each edge e of the tree the server caches the pre-split weight-updates Δ_e of all clients belonging to the two different sub-branches. This way the new client can be moved down the tree along the path of highest similarity.

- If the cluster $c_{v_{child}}$ was created by bi-partitioning the cluster $c_{v_{parent}}$ in CFL then the nodes v_{parent} and v_{child} are connected via a directed edge $e \in E$
- At every edge $e(v_{parent} \rightarrow v_{child})$ the pre-split weight-updates of the children clients

$$\Delta_e = \{\text{SGD}(\theta^*_{v_{parent}}, D_i) - \theta_{v^*_{parent}} | i \in c_{v_{child}}\} \tag{6.42}$$

  are cached

An exemplary parameter tree is shown in Figure 6.5. When a new client joins the training it can get assigned to a leaf cluster by iteratively traversing the parameter tree from the root to a leaf, always moving to the branch which contains the more similar client updates according to Algorithm 13.

Another feature of building a parameter tree is that it allows the server to provide every client with *multiple* models at varying specificity. On the path from root to leaf, the models get more specialized with the most general model being the FL model at the root. Depending on application and context, a CFL client could switch between models of different generality. Furthermore a parameter tree allows us to ensemble multiple models of different specificity together. We believe that investigations along those lines are a promising direction of future research.

Putting all pieces from the previous sections together, we arrive at a protocol for general privacy-preserving CFL which is described in Algorithm 12

6.5 Practical Considerations

In section 6.2.1 we showed that the cosine similarity criterion does distinguish different incongruent clients under three conditions: (a) Federated Learning has converged

to a stationary point θ^*, (b) every client holds enough data s.t. the empirical risk approximates the true risk, (c) cosine similarity is computed between the full gradients of the empirical risk. In this section we will demonstrate that in practical problems none of these conditions have to be fully satisfied. Instead, we will find that CFL is able to correctly infer the clustering structure even if clients only hold small data sets and are trained to an approximately stationary solution of the Federated Learning objective. Furthermore we will see that cosine similarity can be computed between weight-updates instead of full gradients, which even improves performance.

In the experiments of this section we consider the following Federated Learning setup: All experiments are performed on either the MNIST (LeCun, 1998) or CIFAR-10 (Krizhevsky, Nair, and Hinton, 2014) data set using $M = 20$ clients, each of which belonging to one of $K = 4$ clusters. Every client is assigned an equally sized random subset of the total training data. To simulate an incongruent clustering structure, every clients' data is then modified by randomly swapping out two labels, depending on which cluster a client belongs to. For example, in all clients belonging to the first cluster, data points labeled as "1" could be relabeled as "7" and vice versa, in all clients belonging to the second cluster "3" and "5" could be switched out in the same way,

Algorithm 12 Clustered Federated Learning with Privacy Preservation and Weight-Updates

input: initial parameters θ_0, splitting parameters $\varepsilon_1, \varepsilon_2 > 0$, empirical risk approximation error bound $\gamma_{max} \in [0, 1)$, number of local iterations/ epochs n
outout: improved parameters on every client θ_i
init: set initial clusters $\mathcal{C} = \{\{1, .., M\}\}$, set initial models $\theta_i \leftarrow \theta_0 \; \forall i = 1, .., m$, set initial update $\Delta\theta_c \leftarrow 0 \; \forall c \in \mathcal{C}$, clients exchange random seed to create permutation operator P (optional, otherwise set P to be the identity mapping)

while not converged **do**
  for $i = 1, .., M$ **in parallel do**
    Client i does:
    $\theta_i \leftarrow \theta_i + P^{-1}\Delta\theta_{c(i)}$
    $\Delta\theta_i \leftarrow P(\text{SGD}(\theta_i, D_i) - \theta_i)$
  end for
  Server does:
  $\mathcal{C}_{tmp} \leftarrow \mathcal{C}$
  for $c \in \mathcal{C}$ **do**
    $\Delta\theta_c \leftarrow \frac{1}{|c|}\sum_{i \in c} \Delta\theta_i$
    if $\|\Delta\theta_c\| < \varepsilon_1$ **and** $\max_{i \in c} \|\Delta\theta_i\| > \varepsilon_2$ **then**
      $\alpha_{i,j} \leftarrow \frac{\langle \Delta\theta_i, \Delta\theta_j \rangle}{\|\Delta\theta_i\| \|\Delta\theta_j\|}$
      $c_1, c_2 \leftarrow \arg\min_{c_1 \dot\cup c_2 = c}(\max_{i \in c_1, j \in c_2} \alpha_{i,j})$
      $\alpha_{cross}^{max} \leftarrow \max_{i \in c_1, j \in c_2} \alpha_{i,j}$
      if $\alpha_{cross} < \alpha_{cross}^{thresh}$ **then**
        $\mathcal{C}_{tmp} \leftarrow (\mathcal{C}_{tmp} \setminus c) \cup c_1 \cup c_2$
      end if
    end if
  end for
  $\mathcal{C} \leftarrow \mathcal{C}_{tmp}$
end while
return θ

Algorithm 13 Assigning new Clients to a Cluster

Input: new client with data D_{new}, parameter tree $T = (V, E)$
$v \leftarrow v_{root}$
Output: Cluster assignment c_v and stationary cluster model θ_v^*

while $|\text{Children}(v)| > 0$ **do**
  $v_0, v_1 \leftarrow \text{Children}(v)$
  $\Delta\theta_{new} \leftarrow$
    $\text{SGD}(\theta_v^*, D_{new}) - \theta_v^*$
  $\alpha_0 \leftarrow$
    $\max_{\Delta\theta \in \Delta_{(v \to v_1)}} \alpha(\Delta\theta_{new}, \Delta\theta)$
  $\alpha_1 \leftarrow$
    $\max_{\Delta\theta \in \Delta_{(v \to v_2)}} \alpha(\Delta\theta_{new}, \Delta\theta)$
  if $\alpha_0 > \alpha_1$ **then**
    $v \leftarrow v_0$
  else
    $v \leftarrow v_1$
  end if
end while
return c_v, θ_v^*

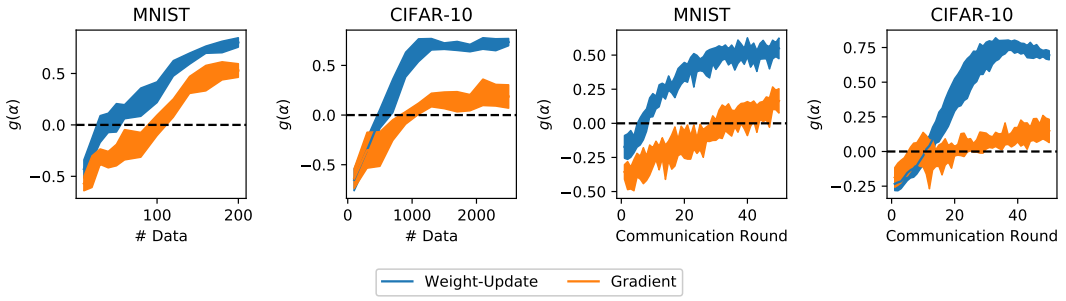


FIGURE 6.6: Panels 1, 2: Separation gap $g(\alpha)$ as a function of the number of data points on every client for the label-swap problem on MNIST and CIFAR. From Corollary 2 we know that CFL will always find a correct bi-partitioning if $g(\alpha) > 0$. On MNIST this is already satisfied if clients hold as little as 20 data points if weight-updates are used for the computation of the similarity α. Panels 3, 4: Separation gap $g(\alpha)$ as a function of the number of communication rounds for the label-swap problem on MNIST and CIFAR. The separation quality monotonically increases with the number of communication rounds of Federated Learning. Correct separation in both cases is already achieved after around 10 communication rounds if α is computed using weight-updates.

and so on. This relabeling ensures that both $\varphi(x)$ and $\varphi(y)$ are approximately the same across all clients, but the conditionals $\varphi(y|x)$ diverge between different clusters. We will refer to this as "label-swap augmentation" in the following. In all experiments we train multi-layer convolutional neural networks and adopt a standard Federated Learning strategy with 3 local epochs of training. We report the separation gap (Definition 3)

$$g(\alpha) := \alpha_{intra}^{min} - \alpha_{cross}^{max} \tag{6.43}$$

which according to Corollary 2 tells us whether CFL will correctly bi-partition the clients:

$$g(\alpha) > 0 \Rightarrow \text{"Correct Clustering"} \tag{6.44}$$

Number of Data points: We start out by investigating the effects of data set size on the cosine similarity. We randomly subsample from each client's training data to vary the number of data points on every client between 10 and 200 for MNIST and 100 and 2400 for CIFAR. For every different local data set size we run Federated Learning for 50 communication rounds, after which training progress has come mostly to halt and we can expect to be close to a stationary point. After round 50, we compute the pairwise cosine similarities between the weight-updates and the separation gap $g(\alpha)$. The results are shown in Figure 6.6 (panel 1,2). As expected, $g(\alpha)$ grows monotonically with increasing data set size. On the MNIST problem as little as 20 data points on every client are sufficient to achieve correct bi-partitioning in the sense of Definition 2. On the more difficult CIFAR problem a higher number of around 500 data points is necessary to achieve correct bi-partitioning.

Proximity to Stationary Solution: Next, we investigate the importance of proximity to a stationary point θ^* for the clustering. Under the same setting as in the previous experiment we reduce the number of data points on every client to 100 for MNIST and to 1500 for CIFAR and compute the pairwise cosine similarities and the separation gap after each of the first 50 communication rounds. The results are shown in Figure 6.6 (panel 3, 4). Again, we see that the separation quality monotonically

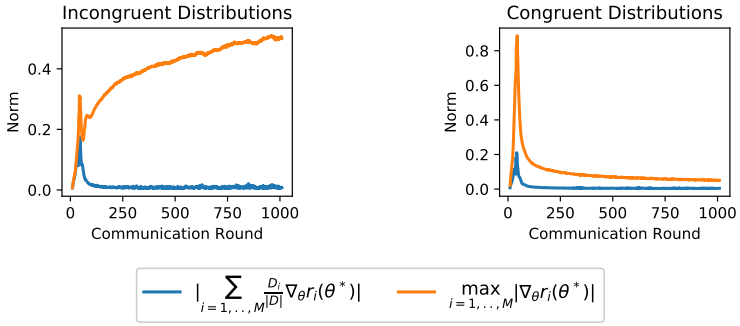


FIGURE 6.7: Experimental verification of the norm criteria (6.29) and (6.28). Displayed is the development of gradient norms over the course of 1000 communication rounds of Federated Learning with two clients holding data from incongruent (left) and congruent distributions (right). In both cases Federated Learning converges to a stationary point of $F(\theta)$ and the average update norm (6.28) goes to zero. In the congruent case the maximum norm of the client updates (6.29) decreases along with the server update norm, while in contrast in the incongruent case it stagnates and even increases.

increases with the number of communication rounds. On MNIST and CIFAR as little as 10 communication rounds are necessary to obtain a correct clustering.

Weight-Updates instead of Gradients: In both the above experiments we computed the cosine similarities α based on either the full gradients

$$\alpha_{i,j} = \frac{\langle \nabla_\theta r_i(\theta), \nabla_\theta r_j(\theta) \rangle}{\|\nabla_\theta r_i(\theta)\| \|\nabla_\theta r_j(\theta)\|} \quad \text{("Gradient")} \tag{6.45}$$

or federated weight-updates

$$\alpha_{i,j} = \frac{\langle \Delta\theta_i, \Delta\theta_j \rangle}{\|\Delta\theta_i\| \|\Delta\theta_j\|} \quad \text{("Weight-Update")} \tag{6.46}$$

over 3 epochs. Interestingly, weight-updates seem to provide even better separation $g(\alpha)$ with fewer data points and at a greater distance to a stationary solution. This comes in very handy as it allows us to leave the Federated Learning communication protocol unchanged. In all following experiments we will compute cosine similarities based on weight-updates instead of gradients.

Distinguishing Congruent and Incongruent Clients: Next, we will experimentally verify the validity of the clustering criteria (6.28) and (6.29) in a Federated Learning experiment on MNIST with two clients holding data from incongruent and congruent distributions. In the congruent case client one holds all training digits "0" to "4" and client two holds all training digits "5" to "9". In the incongruent case, both clients hold a random subset of the training data, but the distributions are modified according to the "label swap" rule described above. Figure 6.7 shows the development of the average update norm (equation (6.28)) and the maximum client norm (equation (6.29)) over the course of 1000 communication rounds. As predicted by the theory, in the congruent case the maximum client norm converges to zero, while in the incongruent case it stagnates and even increases over time. In both cases the average update norm tends to zero, indicating convergence to a stationary point (see Figure 6.7).

The considerations in this section lead us to the following recommendations regarding the selection of the hyperparameters ε_1 and ε_2:

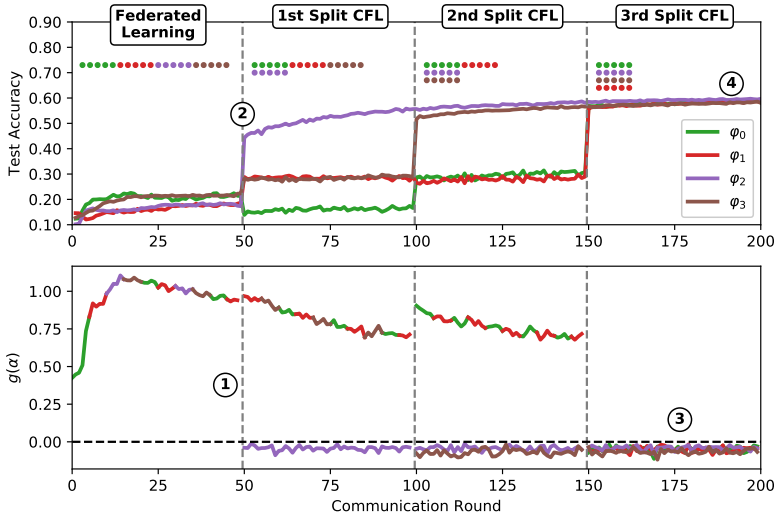


FIGURE 6.8: CFL applied to the "permuted labels problem" on CIFAR with 20 clients and 4 different permutations. Accuracy of the trained models on their corresponding validation sets as well as separation gaps $g(\alpha)$ for all clusters over the course of training. After an initial 50 communication rounds a large separation gap has developed and a first split separates out the purple group of clients, which leads to an immediate drastic increase of accuracy for these clients. In communication rounds 100 and 150 this step is repeated until all clients with incongruent distributions have been separated. After the third split, the model accuracy for all clients has more than doubled and the separation gaps in all clusters have dropped to below zero which indicates that the clustering is finalized.

- As the quality of the clustering improves with proximity to a stationary solution (cf. Fig. 6.6, panel 3, 4), the value for ε_1 should be set as small as the run-time restrictions allow. A good rule of thumb is to set it to around a tenth of the maximum average update norm $\varepsilon_1 \approx \max_t \|\Delta\theta_c^t\|/10$.
- The value of ε_2 should be set in accordance with the number of available clients and/ or prior knowledge on the heterogeneity of the client data. The smaller the value of ε_2, the more likely it is that the client population will be separated by CFL. In our experiments, we obtained good results by setting $\varepsilon_2 \in [\varepsilon_1, 10\varepsilon_1]$.

6.6 Evaluating Clustered Federated Learning in Heterogeneous Settings

In this section, we apply CFL as described in Algorithm 12 to different Federated Learning setups, which are inspired by our motivating examples in the introduction. In all experiments, the clients perform 3 epochs of local training at a batch-size of 100 in every communication round.

Image Classification on CIFAR-10: We split the CIFAR-10 training data randomly and evenly among $M = 20$ clients, which we group into $K = 4$ different clusters. All clients belonging to the same cluster apply the same random permutation $P_{c(i)}$ to their labels such that their modified training and test data is given by

$$\hat{D}_i = \{(x, P_{I(i)}(y)) | (x, y) \in D_i\} \tag{6.47}$$

respective

$$\hat{D_i^{test}} = \{(x, P_{I(i)}(y))|(x,y) \in D^{test}\}. \tag{6.48}$$

The clients then jointly train a 5-layer convolutional neural network on the modified data using CFL with 3 epochs of local training at a batch-size of 100. Figure 6.8 (top) shows the joint training progression: In the first 50 communication rounds, all clients train one single model together, following the conventional Federated Learning protocol. After these initial 50 rounds, training has converged to a stationary point of the Federated Learning objective and the client test accuracies stagnate at around 20%. Conventional Federated Learning would be finalized at this point. At the same time, we observe (Figure 6.8, bottom) that a distinct gap $g(\alpha) = \alpha_{intra}^{min} - \alpha_{cross}^{max}$ has developed (1), indicating an underlying clustering structure. In communication round 50 the client population is therefore split up for the first time, which leads to an immediate 25% increase in validation accuracy for all clients belonging to the "purple" cluster which was separated out (2). Splitting is repeated in communication rounds 100 and 150 until all clusters have been separated and $g(\alpha)$ has dropped to below zero in all clusters (3), which indicates that clustering is finalized. At this point the accuracy of all clients has more than doubled the one achieved by the Federated Learning solution and is now at close to 60% (4). This underlines that after standard FL, our novel CFL can detect the necessity for subsequent splitting and clustering which enables arriving at significantly higher performance. In addition, the cluster structure found can potentially be illuminating as it provides interesting insight about the composition of the complex underlying data distribution.

6.7 Adversarial Robustness of Clustered Federated Learning

Numerous studies have shown that regular Federated Learning fails to converge in the presence of faulty and malicious clients (Blanchard, Guerraoui, Stainer, et al., 2017; Mhamdi, Guerraoui, and Rouault, 2018). Generally, one single bad client can compromise the performance of the entire jointly trained model, and negate the training efforts of all other clients. To mitigate this problem, different robust Federated Learning strategies have been proposed in the literature. However, these existing strategies require modifications to the federated communication protocol to be made and are often computationally expensive.

In this section we will explore the application of CFL to these byzantine settings, where a subset of the client population behaves in an explicitly harmful manner. It is easy to see, that these settings can be subsumed under Assumption 2 by declaring one particular cluster of clients $c_{benign} \in \mathcal{C}$ as the "benign" clients and all other clients, which are incongruent to this cluster as "adversarial": $\mathcal{C} = \{c_{benign}\} \cup \mathcal{C}_{adv}$.

CFL for the Byzantine Setting: In principle, the above described Algorithm for CFL does not need to be modified for the byzantine setting. However, as we assume in the byzantine setting that the majority of clients belongs to one single benign cluster and all other clients are considered adversarial, we can save computation effort by excluding all clients from training, which do not belong to the largest cluster, thus adapting the separation rule from

$$\mathcal{C}_{tmp} \leftarrow (\mathcal{C}_{tmp} \setminus c) \cup c_1 \cup c_2 \quad \text{("regular CFL")} \tag{6.49}$$

to

$$\mathcal{C}_{tmp} \leftarrow (\mathcal{C}_{tmp} \setminus c) \cup \{\arg \max_{c \in \{c_1, c_2\}} |c|\} \quad \text{("adversarially robust CFL")}. \tag{6.50}$$

for the byzantine setting.

Adversarial Federated Learning settings can be roughly organized into two groups: In byzantine settings it is assumed that a subset of the client population behaves in an arbitrary and potentially random manner. Clients may divert the training process by modifying their local data, or the communicated parameter updates, but are not able to adapt their attack based on received training state. This setting has been extensively studied and a variety of robust aggregation rules have been proposed which rely on gradient similarity (Blanchard, Guerraoui, Stainer, et al., 2017; Mhamdi, Guerraoui, and Rouault, 2018), geometric median aggregation (Chen, Su, and Xu, 2017), redundant communication (Chen et al., 2018b) or adaptive model quality estimation (Muñoz-González, Co, and Lupu, 2019). While some of these proposed methods offer convergence guarantees in the byzantine setting, they are also expensive in terms of computation or communication and often require modifications to the federated communication protocol. A more difficult problem setting is that of Federated Learning poisoning. In this setting, (possibly multiple) clients try to introduce a hidden back-door functionality into the jointly trained model (Muñoz-González et al., 2017; Fung, Yoon, and Beschastnikh, 2018; Bagdasaryan et al., 2020; Bhagoji et al., 2019). As the adversaries in this setting are allowed to adapt their attacks based on the model updates they receive from the server, they are much harder to detect and to this day no efficient defense strategies have been proposed. We will thus only focus on byzantine settings in this study.

6.7.1 Evaluating Clustered Federated Learning in Adversarial Settings

We adopt the experimental setup from (Muñoz-González, Co, and Lupu, 2019) to investigate the robustness of Clustered Federated Learning against byzantine and adversarial clients. We perform experiments on the well-known MNIST, Fashion-MNIST and CIFAR-10 data sets on which we train convolutional deep neural networks using SGD with a batch-size of 100. We consider a Federated Learning setting with 100 participating clients among which we split the training data randomly and evenly. In each experiment we declare 30% of the client population to be faulty/ malicious. We consider three different scenarios:

1.) In the **Byzantine** scenario, the malicious clients draw their weight-updates $\Delta\theta$ from a centered Gaussian distribution with isotropic covariance matrix and standard deviation 1 (instead of computing them using stochastic gradient descent). This scenario simulates an undirected attack against the collaborative training procedure.

2.) In the **Label-Flip** scenario, all the labels of the training data for the malicious clients are set to zero. This scenario simulates a directed attack, with the goal to disproportionally bias the jointly trained model towards one specific class.

3.) In the **Noisy** scenario, the training data on the faulty clients is modified by adding independent uniform noise $\hat{x} = x + \mathcal{U}(-10, 10)$ to all pixels and channels. This scenario simulates unstructured noisy distortions of the training data.

4.) In the **Clean** scenario, no adversaries are present. A good robust training algorithm should not harm the convergence in this setting where all clients are benign.

TABLE 6.2: Accuracy achieved by conventional Federated Learning and CFL in the four investigated scenarios. Best performing methods are highlighted in bold face.

		Byzantine	Noisy	Label-Flip	Clean
MNIST	FL	9.8%	96.9%	91.3%	**97.5**
	CFL	**93.19%**	**97.4%**	**97.4%**	97.4%
Fashion-MNIST	FL	9.6%	77.12%	60.6%	79.9
	CFL	**78.0%**	**79.7%**	**79.7%**	**80.2**
CIFAR	FL	10.0%	70.4%	40.1	**76.0**
	CFL	**61.7%**	**74.6%**	**74.7%**	75.3%

In each scenario, we perform Clustered Federated Learning according to Algorithm 12 over the entire client population and set the threshold for the necessary cosine dissimilarity between different clusters to $\alpha_{cross}^{thresh} = 0.02$.

Our goal is to investigate whether CFL as defined in Algorithm 12 is able to detect the faulty and malicious clients in the above scenarios and remove them from the main cluster. Figure 6.10 shows the development of α_{cross} over the first 200 communication rounds for 70 benign and 30 malicious clients on our three data sets and three different scenarios. Every time the value of α_{cross} falls below $\alpha_{cross}^{thresh} = 0.02$ the clients are separated into two different groups according to the rule defined in (6.34). These events are marked in the plot as follows: Whenever adversarial clients are separated from the main cluster of benign clients this is marked green in the plot and the number of adversarial clients that were separated in this particular round. Whenever benign clients are removed from the main cluster this is marked red in the plot.

As we can see, on all data sets and different adversarial scenarios it takes less than 40 communication rounds for all malicious clients to be removed from training. In the noisy and label-flip scenario the adversarial clients all share the same data generating distribution which causes them to be separated out all at once by CFL. In the byzantine scenario, where the adversarial clients communicate Gaussian random updates, every adversary forms it's own cluster and hence it takes several clustering rounds to filter out all adversaries. In the clean scenario the cross-cluster similarity α_{cross} never falls below the similarity threshold. Consequently the client population is never separated and CFL does not harm the convergence in this situation.

On the noisy and label-flip problem on CIFAR the cross cluster similarity falls below the threshold α_{cross}^{thresh} after around 170 communication rounds, which causes benign clients to be separated from the main cluster (marked red). While this has no immediate negative effect on the model accuracy, it should of course be avoided in practice. A possible remedy could be to lower the value of α_{cross}^{thresh} as the training progresses.

In every performed experiment however all of the malicious clients are separated out first and fully. The final accuracy achieved by FL and CFL after 200 rounds of communication is shown in Table 6.2 and the training curves are given in Figure 6.9. As we can see CFL achieves significantly higher accuracy than regular FL in the adversarial scenarios, closely matching the clean baseline, with the accuracy difference being the largest in the byzantine setting where conventional FL diverges.

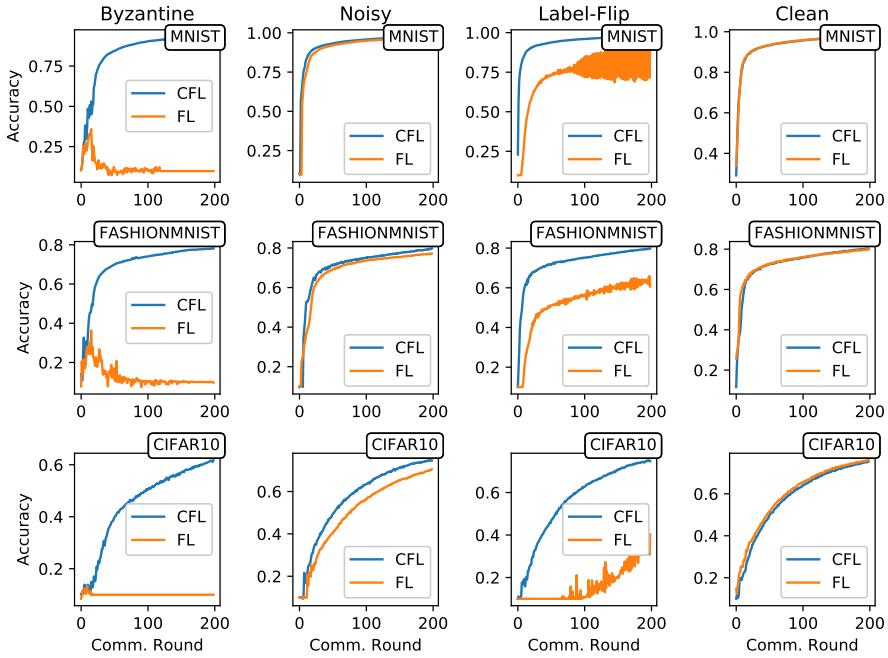


FIGURE 6.9: Development of the cluster candidate dissimilarity α_{cross} for four different adversarial scenarios and three different data sets.

6.8 Summary & Limitations

In this chapter we presented Clustered Federated Learning, a framework for Federated Multi-Task Learning that can improve any existing Federated Learning framework by enabling the participating clients to learn more specialized models. Clustered Federated Learning makes use of our theoretical finding, that at any stationary solution of the Federated Learning objective the cosine similarity between the weight-updates of different clients is highly indicative of the similarity of their data distributions. This crucial insight allows us to provide strong mathematical guarantees on the clustering quality under mild assumptions on the clients and their data, even for arbitrary non-convex objectives.

We demonstrated that CFL can be implemented in a privacy preserving way and without having to modify the FL communication protocol. Moreover, CFL is able to distinguish situations in which a single model can be learned from the clients' data from those in which this is not possible and only separates clients in the latter case. Our experiments on convolutional and recurrent deep neural networks show that CFL can achieve drastic improvements over the Federated Learning baseline in terms of classification accuracy / perplexity in situations where the clients' data exhibits a clustering structure.

We also investigated the application of CFL to byzantine scenarios which can be viewed as a special case of divergent client data distribution where one particular distribution is declared to be the benign distribution and all other distributions are considered adversarial. In experiments on three different data sets and with three different types of adversarial scenarios we find that CFL (without any modifications) is capable of filtering out adversarial clients within relatively few communication rounds. This demonstrates that CFL can offer significant advantages over regular Federated Learning even in situations where clients do not form an obvious clustering structure.

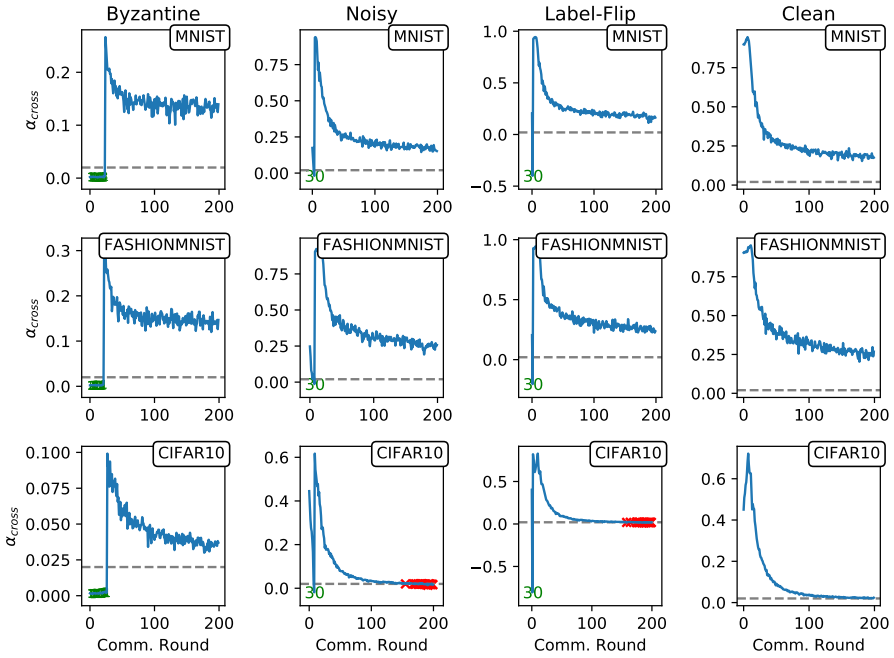


FIGURE 6.10: Development of the cluster candidate dissimilarity α_{cross} for four different adversarial scenarios and three different data sets. Whenever the value of α_{cross} dips below $\alpha_{cross}^{thresh} = 0.02$ the main cluster is separated into two according to equation (6.50). A correct clustering is displayed in greed and incorrect clustering is displayed in red. If the clustering is correct the number of correctly separated clients is shown in the plot. On all three data sets and adversarial scenarios all 30 adversaries are separated out within at most 34 communication rounds. ©2020 IEEE

A challenging setting, which we have not considered in this study is that of multi-modal data heterogeneity, which may arise for instance when client data is heterogeneous *and* some clients behave adversarially. Future research should also address more complex sources of heterogeneity, such as concept drift, where the underlying data-generation process changes overtime. Another extension, which could be worth exploring, is the use of ensemble techniques for re-combining several specialized cluster models obtained by CFL into a mixture of experts.

Finally, we note that our work also exposes a new privacy issue in Federated Learning as it demonstrates that information about client data similarity can be inferred from their weight updates. We argue that the privacy loss inflicted is tolerable in most situations as, without additional knowledge, the mere knowledge of client similarity doesn't reveal anything about the clients' data. Nevertheless this fact should be considered, when implementing Federated Learning for privacy sensitive applications.

Lessons Learned

- At any stationary solution of the Federated Learning objective, the cosine similarity between the weight-updates of different clients is highly indicative of the similarity of their data distributions.
- Based on this criterion, a top-down clustering algorithm can be derived, which organizes clients into groups of similar data.
- Client similarity can be computed in a privacy preserving way by applying a random orthonormal transformation to the parameter updates prior to communication.
- The mean and maximum client update norm can act as stopping criteria for the recursive refinement of client clusters.
- The hierarchical clustering approach provides clients with models of varying degree of personalization.
- Adversarial clients in Federated Learning can be viewed as a special case of deviating client data distributions and are handled automatically by the above described clustering process.

Chapter 7

Federated Learning with Auxiliary Data

While the client data in Federated Learning is typically assumed to be private, in most real-world applications the server additionally has access to unlabeled *auxiliary* data, which roughly matches the distribution of the client data. For instance, for many federated computer vision and natural language processing problems, such auxiliary data can be given in the form of public data bases such as ImageNet (Deng et al., 2009) or WikiText (Merity et al., 2017). These data bases contain millions to billions of data samples but are typically lacking the necessary label information to be useful for training task-specific models. Such auxiliary sources of unlabeled data are exploited in Federated Distillation, where predictions on the unlabeled data are used to transfer the client knowledge and improve model fusion. In this chapter, we explore this core assumption made by all Federated Distillation algorithms and aim to derive maximum utility from the available unlabeled auxiliary data.

Concretely we will demonstrate that a wide range of (out-of-distribution) auxiliary data sets are suitable for self-supervised pre-training which can drastically improve FL performance across different baselines. Exploiting auxiliary data in the model fusion step, we will also propose a novel certainty-weighted Federated Distillation technique, that improves performance of FD on non-iid data substantially, addressing a long-standing problem in FL research.

As we will see, these performance improvements, which almost fully close the performance gap between centralized and distributed training, are possible a) under the same assumptions made in the FD literature, b) with only negligible additional computational overhead for the resource-constrained clients and c) with small quantifiable excess privacy loss.

This chapter is based on

- Felix Sattler, Tim Korjakow, Roman Rischke, and Wojciech Samek (2021b). "FedAUX: Leveraging Unlabeled Auxiliary Data in Federated Learning". In: *CoRR* abs/2102.02514. URL: https://arxiv.org/abs/2102.02514

7.1 Exploiting Auxiliary Data in Federated Learning

In this section, we describe our method for efficient Federated Learning in the presence of unlabeled auxiliary data (FEDAUX). An illustration of our proposed approach is given in Figure 7.1. We first describe FEDAUX for the homogeneous setting were all clients hold the same model prototype. The detailed algorithm for the more general model-heterogeneous setting can be found in Section 7.2. An exhaustive *qualitative* comparison between FEDAUX and baseline methods is given in Section 7.5.

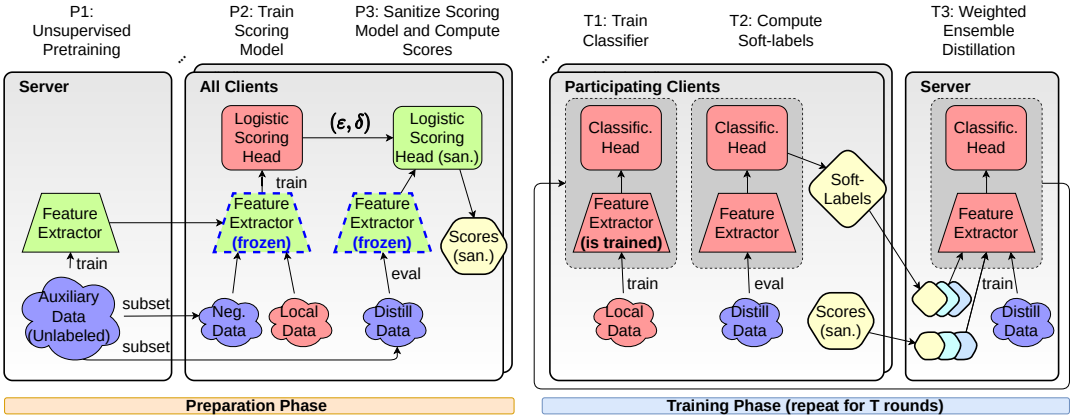


FIGURE 7.1: Illustration of the training procedure of FEDAUX. **Preparation phase:** P1) The unlabeled auxiliary data is used to pre-train a feature extractor (e.g. using contrastive representation learning). P2) The feature-extractor is sent to the clients, where it is used to initialize the client models. Based on extracted features, a logistic scoring head is trained to distinguish local client data from a subset of the auxiliary data. P3) The trained scoring head is sanitized using a (ε, δ)-differentially private mechanism and then used to compute certainty scores on the distillation data. **Training Phase:** T1) In each communication round, a subset of the client population is selected for training. Each selected client downloads a model initialization from the server, and then updates the full model f_i (feature extractor & scoring head) using their private local data. T2) The locally trained classifier and scoring models f_i and s_i are sent to the server, where they are combined into a weighted ensemble. T3) Using the unlabeled auxiliary data and the weighted ensemble as a teacher, the server distills a student model which is used as the initialization point for the next round of federated training.

7.1.1 Problem Setting

We assume the conventional FL setting where a population of n clients is holding potentially non-iid subsets of private labeled data $D_1, .., D_n$, from a training data distribution $(\bigcup_{i \leq n} D_i) \sim \varphi(\mathcal{X}, \mathcal{Y})$. We further make the assumption that the server and the clients both have access to a public collection of unlabeled auxiliary data from a deviating distribution $D_{aux} \sim \psi(\mathcal{X})$. The latter assumption is common to all studies on FD.

One round of federated training is then performed as follows: A subset $\mathcal{S}_t$ of the client population is selected by the server and downloads a model initialization. Starting from this model initialization, each client then proceeds to train a model f_i on it's local private data D_i by taking multiple steps of stochastic gradient descent. We assume that these local models can be decomposed into a feature extractor h_i and a classification head g_i according to $f_i = g_i \circ h_i$. Finally, the updated models f_i, $i \in \mathcal{S}_t$ are sent back to the server, where they are aggregated to form a new server model f, which is used as the initialization point for the next round of FL. The goal of FL is to obtain a server model f, which optimally generalizes to new samples from the training data distribution φ, within a minimum number of communication rounds $t \leq T$.

7.1.2 Federated Ensemble Distillation

As we have learned in Chapters 2 and 5, Federated Ensemble Distillation is a novel method for aggregating the knowledge of FL clients. Instead of aggregating the

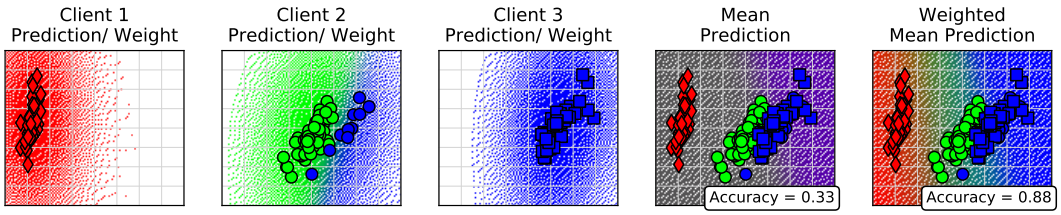


FIGURE 7.2: Weighted Ensemble Distillation illustrated in a toy example on the Iris data set (data points have been projected to their two principal components). Three Federated Learning clients hold disjoint non-iid subsets of the training data. Panels 1-3: Predictions made by linear classifiers trained on the data of each client. Labels and predictions are color-coded, client certainty (measured via Gaussian KDE) is visualized via the alpha-channel. The mean of client predictions (panel 4) only poorly captures the distribution of training data. In contrast, the certainty-weighted mean of client predictions (panel 5) achieves much higher accuracy.

parameters of the client models (e.g. via an averaging operation), a student model is trained on the combined predictions of the clients on some public auxiliary data. Let $x \in D_{aux}$ be a batch of data from the auxiliary distillation data set. Then one iteration of student distillation is performed as

$$\theta^{t,j+1} \leftarrow \theta^{t,j} - \eta \frac{\partial D_{KL}(\mathcal{A}(\{f_i(x)|i \in \mathcal{S}_t\}), \sigma(f(x, \theta^{t,j})))}{\partial \theta^{t,j}}. \tag{7.1}$$

Hereby, D_{KL} denotes the Kullback-Leibler divergence, $\eta > 0$ is the learning rate, σ is the softmax-function and $\mathcal{A}$ is a mechanism to aggregate the soft-labels. Existing work (Lin et al., 2020b) aggregates the client predictions by taking the mean according to

$$\mathcal{A}_{mean}(\{f_i(x)|i \in \mathcal{S}_t\}) = \sigma\left(\frac{\sum_{i \in \mathcal{S}_t} f_i(x)}{|\mathcal{S}_t|}\right), \tag{7.2}$$

and has demonstrated that Federated Ensemble Distillation can outperform parameter averaging based techniques.

7.1.3 Self-supervised Pre-training

Self-supervised representation learning can leverage large records of unlabeled data to create models which extract meaningful features. For the two types of data considered in this study - image and sequence data - strong self-supervised training algorithms are known in the form of contrastive representation learning (Chen et al., 2020b; Wang and Isola, 2020) and next-token prediction (Devlin et al., 2019; Radford et al., 2019). As part of the FEDAUX preparation phase (cf. Fig. 7.1, P1) we propose to perform self-supervised training on the auxiliary data D_{aux} at the server. We emphasize that this step makes no assumptions on the similarity between the local training data and the auxiliary data. This results in a parametrization for the feature extractor h_0. Since the training is performed at the server, using publicly available data, this step inflicts neither computational overhead nor privacy loss on the resource-constrained clients.

7.1.4 Weighted Ensemble Distillation

Different studies have shown that both the training speed, stability and maximum achievable accuracy in existing FL algorithms deteriorate if the training data is

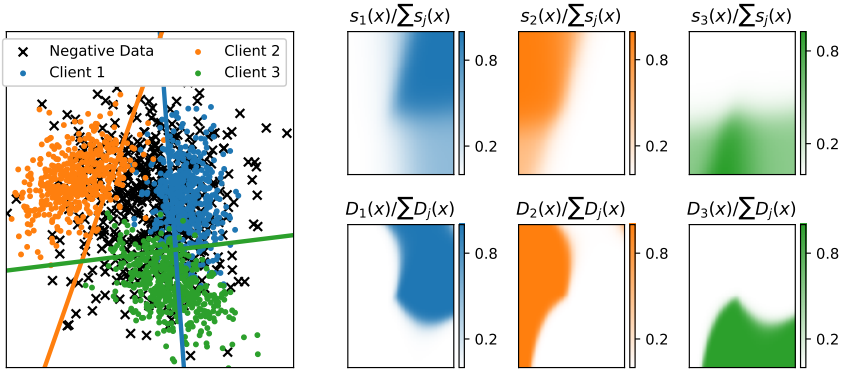


FIGURE 7.3: Left: Toy example with 3 clients holding data sampled from multivariate Gaussian distributions D_1, D_2 and D_3. All clients solve optimization problem J by contrasting their local data with the public negative data, to obtain scoring models s_1, s_2, s_3 respectively. As can be seen in the plots to the right, our proposed scoring method approximates the robust weights proposed in (Mansour, Mohri, and Rostamizadeh, 2008) as it holds $s_i(x)/\sum_j s_j(x) \approx D_i(x)/\sum_j D_j(x)$ on the support of the data distributions.

distributed in a heterogeneous "non-iid" way among the clients (Zhao et al., 2018; Sattler et al., 2020b; Li et al., 2020b). Federated Ensemble Distillation makes no exception to this rule (Lin et al., 2020b).

The underlying problem of combining hypotheses derived from different source domains has been explored in multiple-source domain adaptation theory (Mansour, Mohri, and Rostamizadeh, 2008; Hoffman, Mohri, and Zhang, 2018), which shows that standard convex combinations of the hypotheses of the clients as done in (Lin et al., 2020b) may perform poorly on the target domain. Instead, a distribution-weighted combination of the local hypotheses f_i, obtained on data distributions D_i, according to

$$\bar{f}(x) = \sum_i \frac{D_i(x)}{\sum_j D_j(x)} f_i(x) \tag{7.3}$$

is shown to be robust (Mansour, Mohri, and Rostamizadeh, 2008; Hoffman, Mohri, and Zhang, 2018). A simple toy example, displayed in Figure 7.2, further illustrates this point: Displayed as scatter points are elements of the Iris data set, projected to their two main PCA components. The training data is distributed among three clients in a non-iid fashion, with the label of each data point being indicated by the marker color in the plot. Overlayed in the background are the predictions of linear classifier models that were trained on the local data of each client. As we can see, the models which were trained on the data of clients 1 and 3, uniformly predict that all inputs belong to the "red" and "blue" class respectively. The predictive power of these models and consequently their value as teachers for model distillation is thus very limited. This is also visualized in panel 4, where the mean prediction of the teacher models is displayed. We can however improve the teacher ensemble quite significantly, if we weight each teachers predictions at every location x by it's certainty $s(x)$ (approximated via Gaussian KDE), illustrated via the alpha channel in panels 1-3. As we can see in panel 5, weighing the ensemble predictions raises the accuracy from 33% to 88% in this particular toy example.

Based on these insights, we propose to modify the aggregation rule of FD (7.2) to a certainty-weighted average:

$$\mathcal{A}_s(\{(f_i(x), s_i(x)) | i \in \mathcal{S}_t\}) = \sigma\left(\frac{\sum_{i\in\mathcal{S}_t} s_i(x) f_i(x)}{\sum_{j\in\mathcal{S}_t} s_j(x)}\right) \tag{7.4}$$

The question remains, how to calculate the certainty scores $s_i(x)$ in a privacy preserving way and for arbitrary high-dimensional data, where simple methods, such as Gaussian KDE used in our toy example, fall victim to the curse of dimensionality. To this end, we propose the following methodology:

We split the available auxiliary data randomly into two disjoint subsets,

$$D^- \cup D_{distill} = D_{aux}, \tag{7.5}$$

the "negative" data and the "distillation" data. Using the pre-trained model h_0 ($\rightarrow$ sec. 7.1.3) as a feature extractor, on each client, we then train a logistic regression classifier to separate the local data D_i from the negatives D^-, by optimizing the following regularized empirical risk minimization problem

$$w_i^* = \arg\min_w J(w, h_0, D_i, D^-) \tag{7.6}$$

with

$$J(w, h_0, D_i, D^-) = a \sum_{x\in D_i\cup D^-} l(t_x \langle w, \tilde{h}_0(x)\rangle) + \lambda R(w). \tag{7.7}$$

Hereby $t_x = 2(\mathbb{1}_{x\in D_i}) - 1 \in [-1, 1]$ defines the binary labels of the separation task, $a = (|D_i| + |D^-|)^{-1}$ is a normalizing factor and $\tilde{h}_0(x) = h_0(x)(\max_{x\in D_i\cup D^-} \|h_0(x)\|)^{-1}$ are the normalized features. We choose $l(z) = \log(1 + \exp(z))$ to be the logistic loss and $R(w) = \frac{1}{2}\|w\|_2^2$ to be the ℓ_2-regularizer. Since J is λ-strongly convex in w, problem (7.6) is uniquely solvable. This step is performed only once on every client, during the preparation phase (cf. Fig. 7.1, P2) and the computational overhead for the clients of solving (7.6) is negligible in comparison to the cost of multiple rounds of training the (deep) model f_i.

Given the solution of the regularized ERM w_i^*, the certainty scores on the distillation data $D_{distill}$ can be obtained via

$$s_i(x) = (1 + \exp(-\langle w_i^*, \tilde{h}_0(x)\rangle))^{-1} + \xi. \tag{7.8}$$

A small additive $\xi > 0$ ensures numerical stability when taking the weighted mean in (7.4). We always set $\xi = 1e - 8$.

While the scores $s_i(x)$ can be estimated using a number of different techniques like density estimation, uncertainty quantification (Oala et al., 2020; Abdar et al., 2021) or outlier detection (Ruff et al., 2018; Ruff et al., 2021), we will now present three distinct motivations for using the logistic regression-based approach described above:

First of all, as illustrated using the toy example given in Figure 7.3, the scores obtained via our proposed logistic regression based approach (7.8) give a good approximation to the distribution weights suggested by domain adaptation theory (Mansour, Mohri, and Rostamizadeh, 2008). As we can see in the panels to the right,

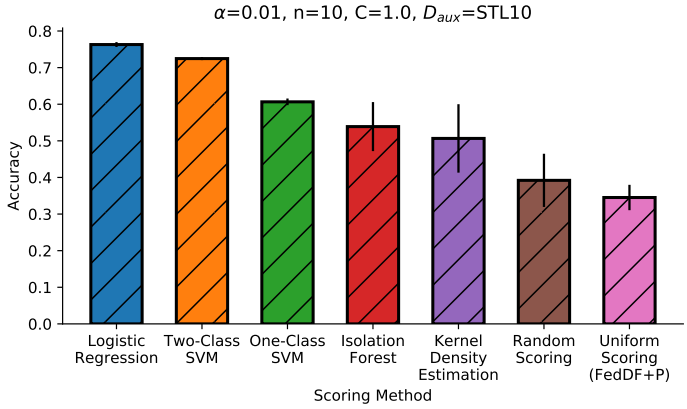


FIGURE 7.4: Comparison of validation performance for Federated Distillation of ResNet-8 on the CIFAR-10 data set when different scoring techniques are used to obtain the certainty weights $s_i(x)$ used during ensemble distillation. Certainty scores obtained via two-class logistic regression achieve the best performance and can readily be augmented with a differentially private mechanism.

it approximately holds

$$\frac{s_i(x)}{\sum_j s_j(x)} \approx \frac{D_i(x)}{\sum_j D_j(x)} \; \forall \; x \in \mathcal{X}, i = 1, .., n \tag{7.9}$$

on the support of the data distributions D_i.

Secondly, scores obtained via logistic regression yield strong empirical performance on highly complex image data. Figure 7.4 shows the maximum accuracy achieved after 10 communication rounds, by different weighted Federated Ensemble Distillation methods in a Federated Learning scenario with 10 clients and highly heterogeneous data ($\alpha = 0.01$, further details on the data splitting strategy are given in Section 7.4). As we can see, the contrastive logistic scoring approach described above distinctively outperforms the uniform scoring approach used in (Lin et al., 2020b) and also yields better results than other generative and discriminative scoring methods, like Gaussian KDE, Isolation Forests or One- and Two-Class SVMs. Details on the implementation of these scoring methods are given in Appendix C.2.

Finally, as we will see in the next section, the logistic scoring mechanism can readily be augmented with differential privacy and provides high utility even under strong formal privacy constraints.

7.1.5 Privacy Analysis

Sharing the certainty scores $\{s_i(x)|x \in D_{distill}\}$ with the central server intuitively causes privacy loss for the clients. After all, a high score $s_i(x)$ indicates, that the public data point $x \in D_{distill}$ is similar to the private data D_i of client i (in the sense of (7.6)). To protect the privacy of the clients, quantify and limit the privacy loss, we propose to use data-level differential privacy (cf. Fig. 7.1, P3). Following the classic definition of (Dwork and Roth, 2014), a randomized mechanism is called differentially private, if it's output on any input data base d is indistinguishable from output on any neighboring database d' which differs from d in one element.

Definition 4. A randomized mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{R}$ satisfies (ε, δ)-differential privacy if for any two adjacent inputs d and d' that differ in only one element and for

any subset of outputs $S \subseteq \mathcal{R}$, it holds that

$$P[\mathcal{M}(d) \in S] \leq \exp(\varepsilon) P[\mathcal{M}(d') \in S] + \delta. \tag{7.10}$$

Differential privacy of a mechanism $\mathcal{M}$ can be achieved, by limiting it's sensitivity

$$\Delta(\mathcal{M}) = \max_{d_1, d_2 \in \mathcal{D}} \|\mathcal{M}(d_1) - \mathcal{M}(d_2)\| \tag{7.11}$$

and then applying a randomized noise mechanism. We adapt a theorem from (Chaudhuri, Monteleoni, and Sarwate, 2011) to establish the sensitivity of (7.6):

Theorem 4. *If $R(\cdot)$ is differentiable and 1-strongly convex and l is differentiable with $|l'(z)| \leq 1\ \forall z$, then the ℓ^2-sensitivity $\Delta_2(\mathcal{M})$ of the mechanism*

$$\mathcal{M} : D_i \mapsto \arg\min_w J(f, h_0, D_i, D^-) \tag{7.12}$$

is at most $2(\lambda(|D_i| + |D^-|))^{-1}$.

The proof can be found in Appendix C.1. As we can see the sensitivity scales inversely with the size of the total data $|D_i| + |D^-|$. From Theorem 4 and application of the Gaussian mechanism (Dwork and Roth, 2014) it follows that the randomized mechanism

$$\mathcal{M}_{san} : D_i \mapsto \arg\min_f J(f, h_0, D_i, D^-) + N \tag{7.13}$$

with $N \sim \mathcal{N}(\mathbf{0}, I\sigma^2)$ and $\sigma^2 = \frac{8 \ln(1.25\delta^{-1})}{\varepsilon^2 \lambda^2 (|D_i| + |D_{aux}|)^2}$ is (ε, δ)-differentially private.

The post-processing property of DP ensures that the release of any number of scores computed using the output of mechanism $\mathcal{M}_{san}$ is still (ε, δ)-private. Note, that in this work we restrict ourselves to the privacy analysis of the scoring mechanism. The differentially private training of deep classifiers f_i is a challenge in it's own right and has been addressed e.g. in (Abadi et al., 2016). Following the basic composition theorem (Dwork and Roth, 2014), the total privacy cost of running FEDAUX is the sum of the privacy loss of the scoring mechanism $\mathcal{M}_{san}$ and the privacy loss of communicating the updated models f_i (the latter is the same for all FL algorithms).

7.2 Algorithm

The training procedure of FEDAUX can be divided into a preparation phase, which is given in Algorithm 14 and a training phase, which is given in Algorithm 15. We describe the general setting where clients may hold different model prototypes P from a set of prototypes $\mathcal{P}$. This general setting simplifies to the setting described in Section 7.1 if $|\mathcal{P}| = 1$.

Preparation Phase (Alg. 14): In the preparation phase, the server uses the unlabeled auxiliary data D_{aux}, to pre-train the feature extractor h^P for each model prototype P using self-supervised training. Suitable methods for self-supervised pre-training are contrastive representation learning (Chen et al., 2020b), or self-supervised language modeling/ next-token prediction (Devlin et al., 2019). The pre-trained feature extractors h_0^P are then communicated to the clients and used to initialize part of the local classifier $f = g \circ h$. The server also communicates the negative data D^- to the clients (in practice we can instead communicate the extracted features $\{|h_0^P(x)|x \in D^-\}$ of the raw data D^- to save communication). Each client then

optimizes the logistic similarity objective J (7.6) and sanitizes the output by adding properly scaled Gaussian noise. Finally, the sanitized scoring model w_i^* is communicated to the server, where it is used to compute certainty scores s_i on the distillation data (the certainty scores can also be computed on the clients, however this results in additional communication of distillation data and scores).

Training Phase (Alg. 15): The training phase is carried out in T communication rounds. In every round $t \leq T$, the server randomly selects a subset $\mathcal{S}_t$ of the overall client population and transmits to them the latest server models $\theta^{\mathcal{R}}[i]$, which match their model prototype P (in round $t = 1$ only the pre-trained feature extractor h_0^P is transmitted). Each selected client updates it's local model by performing multiple steps of stochastic gradient descent (or it's variants) on it's local training data. This results in an updated parameterization θ_i on every client, which is communicated to the server. After all participating clients have finished their local training, the server gathers the updated parameters θ_i. For each model prototype P the corresponding parameters are then aggregated by weighted averaging. Using the model averages as a starting point, for each prototype the server then distills a new model, based on the client's certainty-weighted predictions.

Algorithm 14 FEDAUX Preparation Phase

init: Split $D^- \cup D_{distill} \leftarrow D_{aux}$
init: HashMap $\mathcal{R}$ that maps client i to model prototype P

Server does:
for each model prototype $P \in \mathcal{P}$ **do**
  $h_0^P \leftarrow \text{train\_self\_supervised}(h^P, D_{aux})$
end for
for each client $i \in \{1, .., n\}$ **in parallel do**
  Client i does:
  $P \leftarrow \mathcal{R}[i]$
  $\sigma^2 \leftarrow \frac{8 \ln(1.25\delta^{-1})}{\varepsilon^2 \lambda^2 (|D_i| + |D^-|)^2}$
  $w_i^* \leftarrow \arg\min_w J(w, h_0^P, D_i, D^-) + \mathcal{N}(\mathbf{0}, I\sigma^2)$
  $\gamma_i \leftarrow \max_{x \in D_i \cup D^-} \|h_0^P(x)\|$
end for
Server does:
for $i = 1, .., n$ **do**
  create HashMap
  $s_i \leftarrow \{x \mapsto (1 + \exp(-\langle w_i^*, \gamma_i^{-1} h_0^P(x) \rangle))^{-1} + \xi \text{ for } x \in D_{distill}\}$
end for

Algorithm 15 FEDAUX Training Phase.

init: Training requires feature extractors h_0^P and scores s_i from Alg. 14. Choose learning rate η and set $\xi = 10^{-8}$.
init: HashMap $\mathcal{R}$ that maps client i to model prototype P
init: Inverse HashMap $\tilde{\mathcal{R}}$ that maps model prototype P to set of clients (s.t. $i \in \tilde{\mathcal{R}}[\mathcal{R}[i]] \; \forall i$)
init: Initialize model prototype weights θ^P with feature extractor weights h^P from Alg. 14

for communication round $t = 1, .., T$ **do**
  select subset of clients $\mathcal{S}_t \subseteq \{1, .., n\}$
  for selected clients $i \in \mathcal{S}_t$ **in parallel do**
    Client i does:
    $\theta_i \leftarrow \text{train}(\theta_0 \leftarrow \theta^{\mathcal{R}[i]}, D_i)$
  end for
  Server does:
  for each model prototype $P \in \mathcal{P}$ **do**
    $\theta^P \leftarrow \sum_{i \in \mathcal{S}_t \cap \tilde{\mathcal{R}}[P]} \frac{|D_i|}{\sum_{l \in \mathcal{S}_t \cap \tilde{\mathcal{R}}[P]} |D_l|} \theta_i$
    for mini-batch $x \in D_{distill}$ **do**
      $\tilde{y} \leftarrow \sigma\left(\frac{\sum_{i \in \mathcal{S}_t} s_i[x] f_i(x, \theta_i)}{\sum_{i \in \mathcal{S}_t} s_i[x]}\right)$
      $\theta^P \leftarrow \theta^P - \eta \frac{\partial D_{KL}(\tilde{y}, \sigma(f(x, \theta^P)))}{\partial \theta^P}$
    end for
  end for
end for

7.3 Related Work

Federated Ensemble Distillation: In contrast to centralized model distillation (Hinton, Vinyals, and Dean, 2015), where training and distillation data usually coincide, Federated Distillation makes no restrictions on the auxiliary distillation data and recent work even suggests that useful distillation data can be generated from the teacher models themselves (Nayak et al., 2019). Among existing Federated Distillation techniques, our work is mostly in line with (Lin et al., 2020b; Chen and Chao, 2020) in that it aims to improve overall training performance in FL. Both FEDDF (Lin et al., 2020b) and FEDBE (Chen and Chao, 2020) combine parameter averaging as done in FedAVG (McMahan et al., 2017) with ensemble distillation to improve FL performance. While FEDDF combines client predictions by means of an equally weighted model ensemble, FEDBE forms a Bayesian ensemble from the client models for better robustness to heterogeneous data. Taking FEDDF as a starting point, our proposed FEDAUX algorithm additionally leverages the auxiliary distillation data set for unsupervised pre-training and weights the client predictions in the distillation step according to their prediction certainty to better cope with settings where the client's data generating distributions are statistically heterogeneous.

Weighted Ensembles: Re-weighting the predictions of individual classifiers in an ensemble is a classical technique which has been studied since the '90s with the works of (Hashem and Schmeiser, 1993; Perrone and Cooper, 1993; Sollich and Krogh, 1995). A weighted ensemble of models combines the output of the individual models by means of a weighted average in order to improve the overall generalization performance. The weights allow to indicate the percentage of trust or expected performance for each individual model. See (Sharkey, 1996; Opitz and Maclin, 1999) for an overview of ensemble methods. Instead of giving each client a static weight in the aggregation step of distillation, we weight the clients on an instance base as in (Jiménez, 1998), i.e., each clients prediction is weighted using a data-dependent certainty score. Weighted combinations of weak classifiers are also commonly leveraged in centralized settings in the context of of mixture of experts and boosting methods (Yuksel, Wilson, and Gader, 2012; Masoudnia and Ebrahimpour, 2014; Schapire, 1999).

Unlabeled Data in Federated Learning: To the best of our knowledge, there do not exist any prior studies on the use of unlabeled auxiliary data in FL outside of Federated Distillation methods. Federated semi-supervised learning techniques (Zhang et al., 2020b; Jeong et al., 2020) assume that clients hold both labeled and unlabeled private data from the local training distribution. In contrast, we assume that the server has access to public unlabeled data that may differ in distribution from the local client data. Federated self-supervised representation learning (Zhang et al., 2020a) aims to train a feature extractor on private unlabeled client data. In contrast, we leverage self-supervised representation learning at the server to find a suitable model initialization.

7.4 Experiments

7.4.1 Setup

Datasets and Models: We evaluate FEDAUX and state-of-the-art FL methods on both federated image and text classification problems with large scale convolutional and transformer models respectively. For our image classification problems we train ResNet- (He et al., 2016), MobileNet- (Sandler et al., 2018) and ShuffleNet- (Zhang

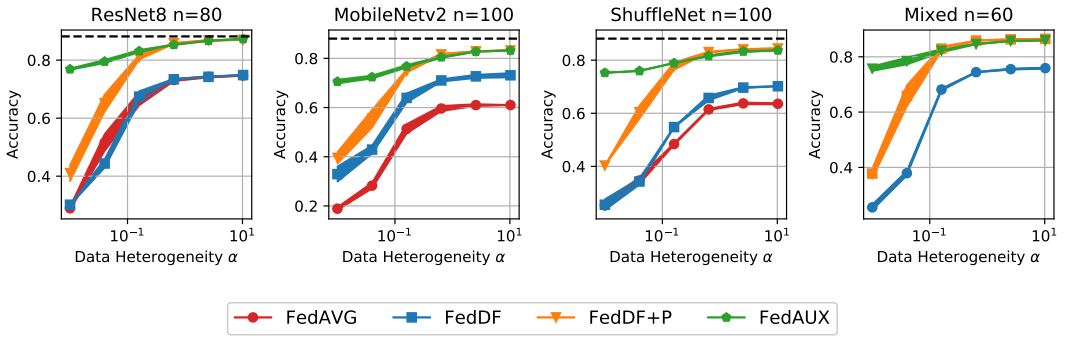


FIGURE 7.5: Evaluation on different neural networks and client population sizes n. Accuracy achieved after $T = 100$ communication rounds by different Federated Distillation methods at different levels of data heterogeneity α. STL-10 is used as auxiliary data set. In the "Mixed" setting one third of the client population each trains on ResNet8, MobileNetv2 and Shufflenet respectively. Black dashed line indicates centralized training performance.

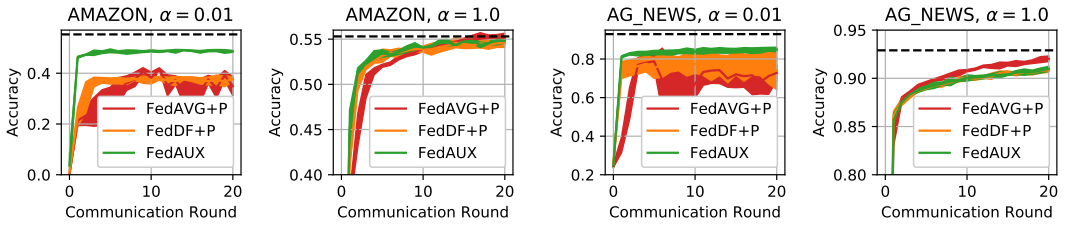


FIGURE 7.6: Evaluating FEDAUX on NLP Benchmarks. Performance of FEDAUX for different combinations of local data sets and heterogeneity levels α. 10 clients training TinyBERT at $\alpha = 0.01$ and $C = 100\%$. Bookcorpus is used as auxiliary data set. Black dashed line indicates centralized training performance.

et al., 2018) type models on CIFAR-10 and CIFAR-100 and use STL-10, CIFAR-100 and SVHN as well as different subsets of ImageNet (Mammals, Birds, Dogs, Devices, Invertebrates, Structures)[1] as auxiliary data. In our experiments, we always use 80% of the auxiliary data as distillation data $D_{distill}$ and 20% as negative data D^-. For our text classification problems we train Tiny-Bert (Jiao et al., 2020) on the AG-NEWS (Zhang, Zhao, and LeCun, 2015b) and Multilingual Amazon Reviews Corpus (Keung et al., 2020) and use BookCorpus (Zhu et al., 2015) as auxiliary data.

Federated Learning Environment and Data Partitioning: We consider Federated Learning problems with up to $n = 100$ participating clients. In all experiments, we split the training data evenly among the clients according to a dirichlet distribution following the procedure outlined in Chapter 2. This allows us to smoothly adapt the level of non-iid-ness in the client data using the Dirichlet parameter α. We experiment with values for α varying between 100.0 and 0.01. A value of $\alpha = 100.0$ results in an almost identical label distribution, while setting $\alpha = 0.01$ results in a split, where the vast majority of data on every client stems from one single class.

Pre-training strategy: For our image classification problems, we use contrastive representation learning as described in (Chen et al., 2020b) for pre-training. We use the default set of data augmentations proposed in the paper and train with the Adam optimizer, learning rate set to 10^{-3} and a batch-size of 512. For our text classification problems, we pre-train using self-supervised next-word prediction.

[1]The methodology for generating these subsets is described in Appendix C.4

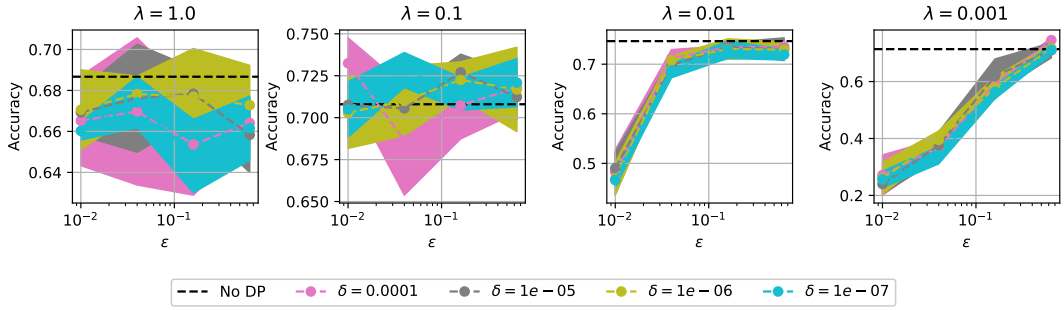


FIGURE 7.7: Performance of FEDAUX for different combinations of the privacy parameters ε, δ and λ. 40 clients training Resnet-8 for $T = 10$ rounds on CIFAR-10 at $\alpha = 0.01$ and $C = 40\%$. STL-10 is used as auxiliary data set.

Training the Scoring model and Privacy Setting: We set the default privacy parameters to $\lambda = 0.1$, $\varepsilon = 0.1$ and $\delta = 1e - 5$ respectively and solve (7.6) by running L-BFGS (Liu and Nocedal, 1989) until convergence (≤ 1000 steps).

Baselines: We compare the performance of FEDAUX to state-of-the-art FL methods: FEDAVG (McMahan et al., 2017), FEDPROX (Li et al., 2020a), Federated Ensemble Distillation (FEDDF) (Lin et al., 2020b) and FEDBE (Chen and Chao, 2020). To clearly discern the performance benefits of the two components of FEDAUX (unsupervised pre-training and weighted ensemble distillation), we also report performance metrics on versions of these methods where the auxiliary data was used to pre-train the feature extractor h ("FEDAVG+P", "FEDPROX+P", "FEDDF+P" resp. "FEDBE+P"). For FEDBE we set the sample size to 10 as suggested in the paper. For FEDPROX we always tune the proximal parameter μ.

Optimization: On all image classification task, we use the very popular Adam optimizer (Kingma and Ba, 2015), with a fixed learning rate of $\eta = 10^{-3}$ and a batch-size of 32 for local training. Distillation is performed for one epoch for all methods using Adam at a batch-size of 128 and fixed learning rate of $5e - 5$. More detailed hyperparameter analysis in Appendix C.6 shows that this choice of optimization parameters is approximately optimal for all of the methods. If not stated otherwise, the number of local epochs E is set to 1.

7.4.2 Evaluating FEDAUX on common Federated Learning Benchmarks

We start out by evaluating the performance of FEDAUX on classic benchmarks for federated image classification. Figure 7.5 shows the maximum accuracy achieved by different Federated Distillation methods after $T = 100$ communication rounds at different levels of data heterogeneity. As we can see, FEDAUX distinctively outperforms FEDDF on the entire range of data heterogeneity levels α on all benchmarks. For instance, when training ResNet8 with $n = 80$ clients at $\alpha = 0.01$, FEDAUX *raises the maximum achieved accuracy from 18.2% to 78.1%* (under the same set of assumptions). The two components of FEDAUX, unsupervised pre-training and weighted ensemble distillation, both contribute independently to the performance improvement, as can be seen when comparing with FEDDF+P, which only uses unsupervised pre-training. Weighted ensemble distillation as done in FEDAUX leads to greater or equal performance than equally weighted distillation (FEDDF+P) across all levels of data heterogeneity. The same overall picture can be observed in the "Mixed" setting where clients train different model architectures. Detailed training curves are given in the Appendix C.3.

TABLE 7.1: Maximum accuracy achieved by FEDAUX and other baseline FL methods after $T = 100$ communication rounds, at different participation rates C and levels of data heterogeneity α. 20 Clients training ResNet-8 on CIFAR-10. Auxiliary data used is STL10. The maximum accuracy per setting is highlighted in bold face.

	$\alpha = 0.01$			$\alpha = 100.0$		
Method	$C = 0.2$	$C = 0.4$	$C = 0.8$	$C = 0.2$	$C = 0.4$	$C = 0.8$
FEDAVG	19.9±0.7	23.6±2.0	28.9±2.0	81.3±0.1	82.2±0.0	82.3±0.1
FEDPROX	28.4±2.5	34.0±1.9	42.0±1.0	81.4±0.1	82.3±0.2	82.0±0.3
FEDDF	25.0±0.8	27.8±0.8	30.6±0.3	80.8±0.1	81.4±0.3	81.5±0.3
FEDBE	20.9±0.6	25.7±1.4	29.1±0.1	81.4±0.7	82.0±0.1	82.2±0.2
FEDAVG+P	30.4±7.9	32.1±2.0	38.4±0.5	89.0±0.1	**89.5±0.1**	**89.6±0.1**
FEDPROX+P	42.8±2.7	43.1±0.2	49.0±0.7	88.9±0.0	89.1±0.1	89.4±0.0
FEDDF+P	28.8±3.0	39.3±3.6	48.1±1.1	88.8±0.0	88.9±0.1	88.9±0.1
FEDBE+P	30.2±2.2	29.8±0.8	37.7±0.0	**89.1±0.1**	89.5±0.2	89.5±0.0
FEDAUX	**54.2±0.3**	**71.2±2.1**	**78.5±0.0**	88.9±0.0	89.0±0.0	89.0±0.1

TABLE 7.2: Maximum accuracy achieved by FEDAUX and other baseline FL methods after 100 communication rounds, when different sets of unlabeled auxiliary data are used for pre-training and/ or distillation. 40 Clients training ResNet-8 on CIFAR-10 at $C = 40\%$. The maximum accuracy per setting is highlighted in bold face.

		Auxiliary Data							
α	Method	STL-10	CIFAR-100	SVHN	Invertebr.	Birds	Devices	Dogs	Structures
0.01	FEDDF	27.9±3.2	29.5±6.2	28.1±3.9	28.5±3.6	30.1±2.0	26.3±0.2	28.9±5.1	30.2±7.0
	FEDDF+P	43.0±5.2	41.6±1.1	29.6±3.4	38.8±6.5	41.4±5.9	35.9±4.9	41.1±7.3	36.7±7.1
	FEDAUX	**76.8±0.9**	**71.5±2.5**	**43.7±1.5**	**68.2±0.7**	**65.7±3.1**	**71.5±0.1**	**71.8±3.8**	**64.1±3.3**
100.00	FEDDF	79.3±0.7	79.9±0.1	80.9±0.1	80.2±0.1	80.2±0.4	79.4±0.3	79.7±0.4	80.1±0.2
	FEDDF+P	88.3±0.0	86.7±0.0	**81.7±0.2**	87.4±0.1	87.6±0.0	87.7±0.1	88.4±0.0	**87.4±0.1**
	FEDAUX	**88.5±0.0**	**86.7±0.1**	81.6±0.0	**87.8±0.1**	**87.8±0.1**	**87.8±0.0**	**88.6±0.0**	87.3±0.1

Table 7.1 compares the performance of FEDAUX and baseline methods at different client participation rates C. We can see that FEDAUX benefits from higher participation rates. In all scenarios, methods which are initialized using the pre-trained feature-extractor h_0 distinctively outperform their randomly initialized counterparts. In the iid setting at $\alpha = 100.0$ FEDAUX is mostly en par with the (improved) parameter averaging based methods FEDAVG+P and FEDPROX+P, with a maximum performance gap of 0.8%. At $\alpha = 0.01$ on the other hand FEDAUX outperforms all other methods with a margin of up to 29%.

7.4.3 Evaluating FEDAUX on NLP Benchmarks

Figure 7.6 shows learning curves for federated training of TinyBERT on the Amazon and AG-News data sets at two different levels of data heterogeneity α. We observe, that FEDAUX significantly outperforms FEDDF+P as well as FEDAVG+P in the heterogeneous setting ($\alpha = 0.01$) and reaches 95% of its final accuracy after one communication round on both data sets, indicating suitability for one-shot learning. On more homogeneous data ($\alpha = 1.0$) FEDAUX performs mostly en par with pre-trained versions of FEDAVG and FEDDF, with a maximal performance gap of 1.1 % accuracy on the test set. We note, that effects of data heterogeneity are less severe as in this setting as both the AG News and the Amazon data set only have four and five labels respectively and an α of 1.0 already leads to a distribution where each clients

TABLE 7.3: One-shot performance of different FL methods. Maximum accuracy achieved after $T = 1$ communication rounds at participation-rate $C = 100\%$. Each client trains for $E = 40$ local epochs. The maximum accuracy per setting is highlighted in bold face.

	MobileNetv2, $n = 100$				Shufflenet, $n = 100$			
Method	$\alpha = 0.01$	$\alpha = 0.04$	$\alpha = 0.16$	$\alpha = 10.24$	$\alpha = 0.01$	$\alpha = 0.04$	$\alpha = 0.16$	$\alpha = 10.24$
FEDAVG	10.3±0.0	13.6±2.3	23.6±0.0	30.5±0.9	12.1±0.8	17.4±0.4	28.2±0.8	37.8±0.7
FEDPROX	11.6±0.8	14.3±1.4	23.7±0.3	30.5±0.5	12.9±1.7	18.9±0.2	29.4±0.3	38.9±0.5
FEDDF	16.8±4.2	29.5±3.8	37.7±1.1	40.4±0.5	16.0±5.1	27.3±0.1	38.7±0.2	45.5±0.5
FEDAVG+P	24.3±1.1	44.0±4.4	57.6±3.7	69.9±0.0	25.5±1.4	44.2±0.1	62.9±1.6	71.9±0.1
FEDPROX+P	27.2±2.2	43.4±3.6	56.9±3.9	70.0±0.1	28.4±0.2	47.1±1.5	63.3±1.2	71.9±0.1
FEDDF+P	46.7±5.6	61.1±1.3	67.6±0.5	71.2±0.1	40.4±2.7	59.4±0.8	68.8±0.2	72.7±0.0
FEDAUX	**64.8±0.0**	**65.5±1.0**	**68.2±0.2**	**71.3±0.1**	**66.9±0.6**	**68.6±0.4**	**70.8±0.3**	**72.9±0.1**

owns a subset of the private data set containing all possible labels. Further details on our implementation can be found the Appendix C.5.

7.4.4 Privacy Analysis of FEDAUX

Figure 7.7 examines the dependence of FEDAUX' training performance of the privacy parameters ε, δ and the regularization parameter λ. As we can see, performance comparable to non-private scoring is achievable at conservative privacy parameters ε, δ. For instance, at $\lambda = 0.01$ setting $\varepsilon = 0.04$ and $\delta = 10^{-6}$ reduces the accuracy from 74.6% to 70.8%. At higher values of λ, better privacy guarantees have an even less harmful effect, at the cost however of an overall degradation in performance. Throughout this empirical study, we have set the default privacy parameters to $\lambda = 0.1$, $\varepsilon = 0.1$ and $\delta = 1e - 5$. We also perform an empirical privacy analysis in the Appendix C.7, which provides additional intuitive understanding and confidence in the privacy properties of our method.

7.4.5 Evaluating the dependence on Auxiliary Data

Next, we investigate the influence of the auxiliary data set D_{aux} on unsupervised pre-training, distillation and weighted distillation respectively. We use CIFAR-10 as training data set and consider 8 different auxiliary data sets, which differ w.r.t their similarity to this client training data - from more similar (STL-10, CIFAR-100) to less similar (Devices, SVHN)[2]. Table 7.2 shows the maximum achieved accuracy after $T = 100$ rounds when each of these data sets is used as auxiliary data. As we can see, performance *always* improves when auxiliary data is used for unsupervised pre-training. Even for the highly dissimilar SVHN data set (which contains images of house numbers) performance of FEDDF+P improves by 1% over FEDDF in both the iid and non-iid regime. For other data sets like Dogs, Birds or Invertebrates performance improves by up to 14%, although they overlap with only one single class of the CIFAR-10 data set. The out-performance of FEDAUX on such a wide variety of highly dissimilar data sets suggest that beneficial auxiliary data should be available in the majority of practical FL problems and also has positive implications from the perspective of privacy. Interestingly, performance of FEDDF seems to only weakly correlate with the performance of FEDDF+P and FEDAUX as a function of the auxiliary data set. This suggests, that the properties, which make a data set useful

[2]The CIFAR-10 data set contains images from the classes airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truc.

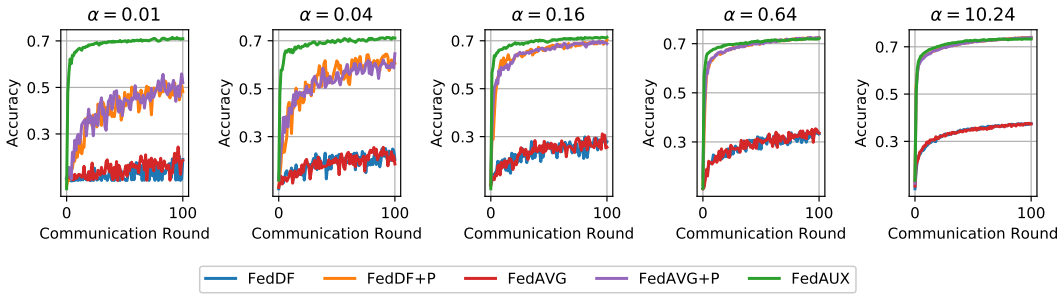


FIGURE 7.8: Linear evaluation of FEDAUX. Training curves for different Federated Learning methods at different levels of data heterogeneity α when only the classification head g is updated in the training phase. A total of $n = 80$ clients training ResNet8 on CIFAR-10 at $C = 40\%$, using STL-10 as auxiliary data set.

for distillation are not the same ones that make it useful for pre-training and weighted distillation. Investigating this relationship further is an interesting direction of future research.

7.4.6 FEDAUX in Hardware-Constrained Settings

Linear Evaluation: In settings where the FL clients are hardware-constrained mobile or IoT devices, local training of entire deep neural networks like ResNet8 might be infeasible. We therefore also consider the evaluation of different FL methods, when only the linear classification head g is updated during the training phase. Figure 7.8 shows training curves in this setting when clients hold data from the CIFAR-10 data set. We see that in this setting performance of FEDAUX is high, independent of the data heterogeneity levels α, suggesting that in the absence of non-convex training dynamics our proposed scoring method actually yields robust weighted ensembles in the sense of (Mansour, Mohri, and Rostamizadeh, 2008). We note, that FEDAUX also trains much more smoothly, than all other baseline methods.

One-Shot Evaluation: In many FL applications, the number of times a client can participate in the federated training is restricted by communication, energy and/ or privacy constraints (Guha, Talwalkar, and Smith, 2019; Papernot et al., 2018). To study these types of settings, we investigate the performance of FEDAUX and other FL methods in federated one-shot learning where we set $T = 1$ and $C = 100\%$. Table 7.3 compares performance in this setting for $n = 100$ clients training MobileNetv2 resp. ShuffleNet. FEDAUX outperforms the baseline methods in this setting at all levels of data heterogeneity α.

7.5 Discussion and Qualitative Comparison with Baselines

The experiments performed in the previous section demonstrate that FEDAUX outperforms state-of-the-art Federated Learning methods by wide margins, in particular if the training data is distributed in a heterogeneous way among the clients. In Table 7.4 we additionally provide a qualitative comparison between FEDAUX and the baseline methods FEDAVG and FEDDF. We can note the following:

Client workload: Compared with FEDAVG and FEDDF, FEDAUX additionally requires the clients to once solve the λ-strongly convex ERM (7.6). For this problem linearly convergent algorithms are known (Liu and Nocedal, 1989) and thus the

TABLE 7.4: Qualitative Comparison of the computational complexity, communication overhead and privacy loss after T communication rounds as well as implicit assumptions made by different Federated Learning methods.

	FEDAVG	FEDDF, FEDBE	FEDAUX (preparation phase)	FEDAUX (training phase)
Operations (Clients)	Local Training ($\times T$)	Local Training ($\times T$)	Solve λ-strongly convex ERM (7.6)	Local Training ($\times T$)
Operations (Server)	Model Averaging ($\times T$)	Model Averaging, Distillation ($\times T$)	Self-Supervised Pre-training of h_0, Computation of certainty scores s_i	Model Averaging, Distillation ($\times T$)
Communication Clients $\rightarrow$ Server	Model Parameters f_i ($\times T$)	Model Parameters f_i ($\times T$)	Scoring Models w_i^*	Model Parameters f_i ($\times T$)
Communication Server $\rightarrow$ Clients	Model Parameters f ($\times T$)	Model Parameters f ($\times T$)	Negative Data D^-, Feature Extractor h_0	Model Parameters f ($\times T$)
Privacy Loss	Privacy loss of communicating f_i ($\times T$)	Privacy loss of communicating f_i ($\times T$)	(ε, δ)-DP	Privacy loss of communicating f_i ($\times T$)
Assumptions	No Assumptions	Auxiliary Data	Auxiliary Data	Auxiliary Data

computational overhead is negligible compared with the complexity of multiple rounds of locally training deep neural networks.

Server workload: FEDAUX also adds computational load to the server for self-supervised pre-training and computation of the certainty scores s_i. As the server is typically assumed to have massively stronger computational resources than the clients, this can be neglected.

Communication Client $\rightarrow$ Server: Once, in the preparation phase of FEDAUX, the scoring models w_i^* need to be communicated from the clients to the server. The overhead of communicating these H-dimensional vectors, where H is the feature dimension, is negligible compared to the communication of the full models f_i.

Communication Server $\rightarrow$ Clients: FEDAUX also requires the communication of the negative data D^- and the feature extractor h_0 from the server to the clients. The overhead of sending h_0 is lower than sending the full model f, and thus the total downstream communication is increased by less than a factor of $(T+1)/T$. The overhead of sending D^- is small (in our experiments $|D^-| = 0.2|D_{aux}|$) and can be further reduced by sending extracted features $\{|h_0^P(x)|x \in D^-\}$ instead of the full data. For instance, in our experiments with ResNet-8 and CIFAR-100 we have $|D^-| = 12000$ and $h_0^P(x) \in \mathbb{R}^{512}$, resulting in a total communication overhead of $12000 \times 512 \times 4B = 24.58$MB for D^-. For comparison the total communication overhead of once sending the parameters of ResNet-8 (needs to be done T times) is 19.79MB.

Privacy Loss: Communicating the scoring models w_i^* incurs additional privacy loss for the clients. Using our proposed sanitation mechanism this process is made (ε, δ)-differentially private. Our experiments in section 7.4.4 demonstrate that FEDAUX can achieve drastic performance improvements, even under conservative privacy constraints. All empirical results reported are obtained with (ε, δ) differential privacy at $\varepsilon = 0.1$ and $\delta = 10^{-5}$.

Assumptions: Finally, FEDAUX makes the additional assumption that unlabeled

auxiliary data is available to the server. This assumption is made by all Federated Distillation methods including FEDDF.

In conclusion, FEDAUX requires comparable resources as state-of-the-art Federated Distillation methods and has similar privacy properties, while at the same time achieving significantly better performance.

7.6 Summary & Limitations

In this chapter, we have explored Federated Learning in the presence of unlabeled auxiliary data, an assumption made in the quickly growing area of Federated Distillation. By leveraging auxiliary data for unsupervised pre-training and certainty weighted ensemble distillation we were able to demonstrate that this assumption is rather strong and can lead to drastically improved performance of FL algorithms. As we have seen, these performance improvements can be obtained even if the distribution of the auxiliary data is highly divergent from the client data distribution and are maintained when the certainty scores are obfuscated using a strong differential privacy mechanism. Additionally, our detailed qualitative comparison with baseline methods revealed that FEDAUX incurs only marginal excess computation and communication overhead.

On a more fundamental note, the dramatic performance improvements observed in FEDAUX call into question the common practice of comparing FD based methods (which assume auxiliary data) with parameter averaging based methods (which do not make this assumption) (Lin et al., 2020b; Chen and Chao, 2020) and thus have implications for the future evaluation of FD methods in general.

We also highlight, that while FEDAUX obfuscates the *certainty scores*, provided by each client, using a differentially private noise mechanism, the local model *parameters* might still reveal details about the private local training data (as they also do with all baseline methods considered in this chapter). A straight-forward extension to our proposed approach which could rectify this issue is to make use of differentially private local training algorithms as described in (Abadi et al., 2016; Mohassel and Zhang, 2017). Finally we note that, while showing very strong empirical performance, the logistic certainty scoring approach presented in this chapter is motivated mainly by heuristic arguments as well as it's amenability to differentially private augmentation. A deeper principled analysis of the weighted ensemble distillation framework presented in this chapter could lead to even better scoring methods, further improving model fusion performance in the presence of heterogeneous data.

Lessons Learned

- The unlabeled auxiliary data necessary to perform Federated Distillation can be used for unsupervised pre-training at the server, which improves performance, without inflicting any computational overhead or privacy loss on the resource-constrained clients.
- Domain adaptation theory (Mansour, Mohri, and Rostamizadeh, 2008) has shown that standard convex combinations of the hypotheses of different learners yield sub-optimal generalization to the target domain, if there is a discrepancy between the individual training data distributions.
- By weighing classifiers during model distillation according to their individual data-specific certainty, this issue can be overcome and Federated Distillation can be made more robust against such heterogeneous distributions of client data.
- Meaningful certainty weights can be obtained in an efficient and privacy-preserving way, by contrasting local client data from auxiliary data in feature space, using a differentially private logistic scoring mechanism.
- Effective weighted ensemble distillation can be achieved with minimal additional workload, communication overhead or privacy loss for the often resource-constrained clients.

Chapter 8

Conclusion

We conclude this thesis with a summary of the presented methods and results, provide an overview over possible future research directions and unanswered questions, and give examples of successive work profiting from the methods presented in this thesis.

8.1 Thesis Summary

In this thesis, we have provided practical solutions to several challenges associated with the distributed training of machine learning models. Our proposed training algorithms and methodologies jointly address issues of communication-efficiency, data and model heterogeneity, personalisation, robustness and privacy, among others.

For three distributed training settings, namely parallel training in the data center, parameter averaging based Federated Learning and Federated Distillation, we have provided communication-efficient optimization frameworks, that are robust to a wide variety of learning environments and constraints. We demonstrated, that our proposed compression methodologies can reduce communication in these settings by several orders of magnitude, without negatively affecting the training performance. This opens up numerous new possibilities for application of distributed training in resource constraint IoT settings and will help the widespread adoption of Federated Learning solutions.

With Clustered Federated Learning, we have developed an entirely new framework for treating data heterogeneity in Federated Learning. We have provided simple to implement and computationally efficient tools, based on the cosine similarity between the local client updates, that provably allow us to detect the hidden clustering structure in a large population of heterogeneous clients, making it possible to automatically separate clients into clusters of jointly trainable data distribution. In extensive studies we demonstrated that CFL can lead to improved training performance in a variety of practical use cases and offers robustness to byzantine scenarios, where some clients may behave in an adversarial way.

Finally, we explored Federated Learning in the presence of unlabeled auxiliary data, an assumption made in the quickly growing area of Federated Distillation. By leveraging auxiliary data for unsupervised pre-training and weighted ensemble distillation we were able to demonstrate that this assumption can lead to drastically improved performance of FL algorithms, for a wide variety of highly dissimilar auxiliary data sets, in many cases fully closing the performance gap to centralized training. Our results constitute a new state-of-the-art in general-purpose federated training of deep machine learning classifiers. At the same time they reveal the limited merit in comparing Federated Distillation based methods with parameter averaging based methods, which do not assume availability of auxiliary data. These results thus have implications for the future evaluation of Federated Distillation methods in general.

8.2 Limitations and Outlook

While the contributions made in this thesis address a number of important challenges in distributed learning in general and Federated Learning in particular, many open problems still remain.

For instance, while our proposed communication-efficient training schemes offer a basic level of privacy by processing data on-device, they do not provide formal privacy guarantees to protect against sophisticated data reconstruction attacks as described for instance in (Hitaj, Ateniese, and Perez-Cruz, 2017). A natural extension of these techniques would therefore constitute of combining them with privacy mechanisms like differential privacy that offer formal guarantees. Recent work (Li et al., 2019) hints at the fact that, if combined correctly, communication-efficiency and privacy can be two sides of the same coin, mutually strengthening each other. This is also in line with intuition, which tells us that limiting the efflux of information from a client to the server should also put restrictions on the privacy loss.

Also often overlooked, when investigating distributed learning in well defined lab conditions, are a number of practical concerns that arise when executing these techniques in real-world production settings, where both the data and the behavior of the participating clients can be arbitrarily ill-behaved. With Clustered Federated Learning we have provided a first preliminary system which is able to treat many of these unexpected failure modes. However issues may still arise in settings were heterogeneity is multi-modal, for instance when client data is heterogeneous *and* some clients behave adversarially. Future research also needs to address more complex sources of heterogeneity, such as concept drift, where the underlying data-generation process changes overtime. Furthermore, there is a great need for cheap and simple tools to diagnose heterogeneity in distributed systems a priori. Such tools could be of great value in helping to calibrate distributed learning algorithms to the particular training conditions.

Another interesting direction of future research is the application of our proposed robust multi-task learning techniques to Federated Distillation settings. As we have demonstrated in this thesis, Federated Distillation techniques have a variety of beneficial properties, with respect to communication-efficiency, training-performance and adversarial robustness. We therefore believe that these methods are here to stay and will influence and shape distributed training efforts for years to come. Currently these methods still require access to unlabeled auxiliary data, which can not be guaranteed in all Federated Learning applications, however techniques have been introduced that enable the generation of synthetic distillation data directly from the teacher models (Nayak et al., 2019).

Furthermore, since the training data is often completely unknown to anyone using models obtained via Federated Learning, these models typically will need to be thoroughly inspected before being deployed in production. In particular, harmful effects from model poisoning (Fang et al., 2020) and backdoor attacks (Bagdasaryan et al., 2020) need to be accounted for. Preliminary defense mechanisms via activation clustering (Chen et al., 2018a) or adversarial training (Zhao et al., 2019) have been proposed, but future work should also explore the use of explainability techniques (Samek et al., 2021; Montavon, Samek, and Müller, 2018) for post-hoc analysis of Federated Learning models in order to improve trustworthiness and robustness.

Lastly, we believe that many of the techniques introduced in this thesis in the context of Federated Learning do not necessarily require a central server and could easily be extended to fully distributed peer-to-peer learning settings.

8.3 Impact

The techniques proposed in this work are already being used to tackle real-world problems in diverse areas such as healthcare, acoustics and IoT. For instance (Nelus, Glitza, and Martin, 2021) use unsupervised Clustered Federated Learning for estimation of source-dominated microphone clusters in acoustic sensor networks. The authors demonstrate that the privacy-aware CFL approach has a significant performance-enhancing effect on a multi-sensor gender recognition task.

The technique is also applied in the medical domain, where (Qayyum et al., 2021) utilize CFL for automatic, multi-modal COVID-19 diagnosis at the edge based on healthcare data. Their CFL-based approach achieves results comparable to the central baseline where the specialized models are trained with central data (each on a different type of COVID-19 imagery), and obtains improvements of 16% resp. 11% in overall F1-Scores over the multi-modal model trained in the conventional Federated Learning setup on X-ray and ultrasound data sets.

Other authors have extended the CFL framework to make it more efficient and adaptive to more complicated client interactions. For instance, (Duan et al., 2020) propose to reduce the computational complexity of computing the cosine similarity matrix in CFL by using a low-rank SVD approximation. Aspects of communication-efficiency in the context of Clustered Federated Learning are considered in (Shlezinger, Rini, and Eldar, 2020) and hierarchical clients relationships are modeled in (Briggs, Fan, and Andras, 2020).

Efficient Federated Learning techniques as proposed in this work (Sattler et al., 2019; Sattler et al., 2020b; Sattler et al., 2021a) are of significant interest to both the research community and practitioners which are trying to build smart distributed IoT applications. Compression techniques such as sparsification, quantization and efficient encoding which we have joitly analyzed in this work are now enabling wireless edge intelligence in IoT applications (Mills, Hu, and Min, 2019; Amiri and Gündüz, 2020b; Amiri and Gündüz, 2020a; Chen et al., 2020a). Some of the compression techniques introduced in this thesis have also been further refined in subsequent work (Shi et al., 2019; Ren, Yu, and Ding, 2020) to adapt to specific network topologies, hardware setups and communication protocols.

Finally, our proposed communication-efficient training methodologies have also influenced recent standardization efforts of the motion picture experts group (MPEG), resulting in test data, evaluation frameworks and results for neural network compression and Federated Learning use cases (ISO/IEC JTC1/SC29/WG11 MPEG2020/m52375, 2020.).

8.4 Concluding

Distributed training methods like Federated Learning were introduced as a response to the desire to harness vast records of distributed data for the training of powerful large-scale deep learning models, without harming the privacy of the involved data donors. When the foundational groundwork for this thesis was laid, Federated Learning was a young and nascent field, with it's landmark paper (McMahan et al., 2017) being published only in 2017. Since then, great progress has been made and the field has gained tremendous interest by the research community and practitioners alike. Major technology companies have already deployed distributed training workflows in production, and a number of startups were founded with the objective of

using distributed machine learning to address privacy and data collection challenges in various industries.

With the methods introduced in this thesis, we have demonstrated that distributed training workflows are viable even in hardware- and network-constrained conditions, can be robust and adaptive to a wide range of heterogeneous and adversarial training conditions and do need to suffer only marginal performance loss when compared to centralized training. We hope that these methods will facilitate the wide-spread adoption of distributed training workflows and help to demonstrate that data protection and privacy do not have to stand in the way of resourceful data analytics and machine learning.

Appendix A

Communication-Efficient Distributed Training

A.1 Proof of Theorem 1

The theorem assumes that (a) the noise n^t is normally distributed at every time-step $n^t \sim \mathcal{N}(0, \sigma^2)$ with the variance being constant in time $\mathbb{V}(n^t) = \sigma^2$ for all $t = 1, .., T$. The theorem further assumes that (b) the noise is (negatively) self-correlated, with the noise process given by $n^1 = N^1$, $n^t = \alpha n^{t-1} + N^t$, with N^t normally distributed and all N^t uncorrelated, $\alpha \in (-1, 0)$.

Under these two assumptions, the theorem states that the variance of the accumulated noise can be bounded by

$$\mathbb{V}(\sum_{t=1}^{T} n^t) \leq \sigma^2(T(1+\alpha)+1). \tag{A.1}$$

Proof. Since

$$\begin{aligned} n^t = \alpha n^{t-1} + N^t &= \alpha(\alpha n^{t-2} + N^{t-1}) + N^t \\ &= \alpha^2 n^{t-2} + \alpha N^{t-1} + N^t \\ &= \alpha^\tau n^{t-\tau} + \sum_{i=0}^{\tau-1} \alpha^i N^{t-i} \end{aligned} \tag{A.2}$$

it holds that

$$\begin{aligned} \operatorname{cov}(n^{t-\tau}, n^t) &= \operatorname{cov}(n^{t-\tau}, \alpha^\tau n^{t-\tau} + \sum_{i=0}^{\tau-1} \alpha^i N^{t-i}) \\ &= \alpha^\tau \sigma^2 + \sum_{i=0}^{\tau-1} \alpha^i \underbrace{\operatorname{cov}(n^{t-\tau}, N^{t-i})}_{=0} = \alpha^\tau \sigma^2 \end{aligned} \tag{A.3}$$

With equation (A.3) it follows that

$$\mathbb{V}(\sum_{t=1}^{T} n^t) = \sum_{t_1=1}^{T}\sum_{t_2=1}^{T} \mathrm{cov}(n^{t_1}, n^{t_2}) \tag{A.4}$$

$$= \underbrace{\sum_{t=1}^{T} \mathrm{cov}(n^t, n^t)}_{T\sigma^2} + 2\underbrace{\sum_{t=1}^{T-1} \mathrm{cov}(n^t, n^{t+1})}_{\alpha(T-1)\sigma^2} + \tag{A.5}$$

$$2\underbrace{\sum_{t=1}^{T-2} \mathrm{cov}(n^t, n^{t+2})}_{\alpha^2(T-2)\sigma^2} + .. + 2\underbrace{\mathrm{cov}(n^1, n^T)}_{\alpha^{T-1}(1)\sigma^2} \tag{A.6}$$

For negatively correlated noise $\alpha \in (-1, 0)$ we can bound this term by

$$\mathbb{V}(\sum_{t=1}^{T} n^t) = \sigma^2(T + 2\sum_{\tau=1}^{T-1} \alpha^\tau (T - \tau)) \tag{A.7}$$

$$= \sigma^2(T + 2\frac{\alpha^{T+1} - \alpha^2 T + \alpha T - \alpha}{(\alpha - 1)^2}) \tag{A.8}$$

$$= \sigma^2(T + 2\underbrace{\frac{(\alpha - \alpha^2)}{(\alpha - 1)^2}}_{\leq \frac{1}{2}\alpha} T + 2\underbrace{\frac{\alpha^{T+1} - \alpha}{(\alpha - 1)^2}}_{\leq \frac{1}{2}}) \tag{A.9}$$

$$\leq \sigma^2(T(1 + \alpha) + 1) \tag{A.10}$$

□

A.2 Encoding and Decoding

To communicate the sparse ternary weight updates from clients to server and back from server to client we only need to transmit the positions of the non-zero elements in every tensor, along with exactly one bit to signify the sign (μ or $-\mu$). As the distances between the non-zero elements of the weight updates $\tilde{\Delta\theta}$ are approximately geometrically distributed for large layer sizes, we can efficiently encode them in an optimal way using the Golomb encoding. The encoding scheme is given in Algorithm 16, while the decoding scheme is given in Algorithm 17.

A.3 Convergence Proofs

We remember the definition of an α-contraction (Stich, Cordonnier, and Jaggi, 2018) which states that an operator $\mathrm{comp} : \mathbb{R}^d \to \mathbb{R}^d$ is called an a α-contraction if it is satisfies the contraction property

$$\mathbb{E}\|x - \mathrm{comp}(x)\|^2 \leq \alpha\|x\|^2, \ \forall x \in \mathbb{R}^d \tag{A.11}$$

for a parameter $0 \leq \alpha < 1$ To prove the convergence of the communication-efficient sparse ternary compression method we only need to prove that the sparse ternary

compression operator satisfies the above property, with

$$\alpha = \frac{\|\text{top}_{pd}(x)\|_1^2}{pd\|x\|_2^2} < 1 \tag{A.12}$$

The convergence result then follows from (Stich, Cordonnier, and Jaggi, 2018).

Proof. Let $k = \max(\lfloor np \rfloor, 1)$ W.l.o.g. let $x = [v, w] \in \mathbb{R}^d$, with $v \in \mathbb{R}^k$, $w \in \mathbb{R}^{d-k}$ and $\min_i |v_i| \geq \max_j |w_j|$. Let us further denote $\mu = \frac{1}{k}\sum_{i=1}^k |v_i|$. Then

$$\mathbb{E}\|\text{STC}_k(x) - x\|^2 = \sum_{i=1}^{k}(|v_i| - \mu)^2 + \sum_{j=k+1}^{d} w_{j-k}^2 \tag{A.13}$$

$$= \sum_{i=1}^{k} v_i^2 + \sum_{j=k+1}^{d} w_{j-k}^2 - k\mu^2 \tag{A.14}$$

$$= \|x\|_2^2 - k\mu^2 \tag{A.15}$$

$$= (1 - \frac{\tilde{k}}{d})\|x\|_2^2 < \|x\|_2^2 \tag{A.16}$$

with

$$\tilde{k} = k\mu^2 \frac{d}{\|x\|_2^2} \tag{A.17}$$

□

Algorithm 16 Golomb Position Encoding

input: sparse tensor ΔW^*, sparsity p
output: binary message msg

$\mathcal{I} \leftarrow \Delta W^*[:]_{\neq 0}$
$\mathbf{b}^* \leftarrow 1 + \lfloor \log_2(\frac{\log(\phi-1)}{\log(1-p)}) \rfloor$
for $i = 1, .., |\mathcal{I}|$ **do**
  $d \leftarrow \mathcal{I}_i - \mathcal{I}_{i-1}$
  $q \leftarrow (d-1) \text{ div } 2^{\mathbf{b}^*}$
  $r \leftarrow (d-1) \bmod 2^{\mathbf{b}^*}$
  msg.add($\underbrace{1, .., 1}_{q \text{ times}}$, 0, binary$_{\mathbf{b}^*}(r)$)
end for
return msg

Algorithm 17 Golomb Position Decoding

input: binary message msg, bitsize $\mathbf{b}^*$, mean value μ
output: sparse tensor ΔW^*
init: $\Delta W^* \leftarrow 0 \in \mathbb{R}^n$

$i \leftarrow 0; q \leftarrow 0; j \leftarrow 0$
while $i <$ size(msg) **do**
  if msg$[i] = 0$ **then**
    $j \leftarrow j + q2^{\mathbf{b}^*} + \text{int}_{\mathbf{b}^*}(\text{msg}[i+1], .., \text{msg}[i+\mathbf{b}^*]) + 1$
    $\Delta W_j^* \leftarrow \mu$
    $q \leftarrow 0; i \leftarrow i + \mathbf{b}^* + 1$
  else
    $q \leftarrow q + 1; i \leftarrow i + 1$
  end if
end while
return ΔW^*

Appendix B

Clustered Federated Learning

B.1 Proving the Separation Theorem

The Separation Theorem makes a statement about the cosine similarities between the gradients of the empirical risk functions $\nabla_\theta r_i(\theta^*)$ and $\nabla_\theta r_j(\theta^*)$, which are noisy approximations of the true risk gradients $\nabla_\theta R_{I(i)}(\theta^*)$, respective $\nabla_\theta R_{I(j)}(\theta^*)$. To simplify the notation let us first re-define

$$v_l = \nabla_\theta R_l(\theta^*), l = 1, .., K \tag{B.1}$$

and

$$X_i = \nabla_\theta r_i(\theta^*) - \nabla_\theta R_{I(i)}(\theta^*), i = 1, .., M \tag{B.2}$$

Figure B.1 shows a possible configuration in $d = 2$ with $K = 3$ different data generating distributions and their corresponding gradients v_1, v_2 and v_3. The empirical risk gradients $X_i + v_{i(i)}$, $i = 1, .., M$ are shown as dashed lines. The maximum angles between gradients from the same data generating distribution are shown green, blue and purple in the plot. Among these, the green angle is the largest one $\sphericalangle^{max}_{intra}$. The plot also shows the optimal bi-partitioning into clusters 1 and 2 and the minimum angle between the gradient updates from any two clients in different clusters $\sphericalangle^{min}_{cross}$ is displayed in red. As long as

$$\sphericalangle^{max}_{intra} < \sphericalangle^{min}_{cross} \tag{B.3}$$

or equivalently

$$\alpha^{min}_{intra} = \cos(\sphericalangle^{max}_{intra}) > \cos(\sphericalangle^{min}_{cross}) = \alpha^{max}_{cross} \tag{B.4}$$

the clustering will always be correct.

The proof of the Theorem can be organized into three separate steps:

- In Lemma 5, we bound the cosine similarity between two noisy approximations of the same vector α^{min}_{intra} from below
- In Lemma 6, we bound the cosine similarity between two noisy approximations of two different vectors from above
- In Lemma 7, we show that every set of vectors that sums to zero can be separated into two groups such that the cosine similarity between any two vectors from separate groups can be bounded from above
- Lemma 6 and 7 together will allow us to bound the cross cluster similarity α^{max}_{cross} from above

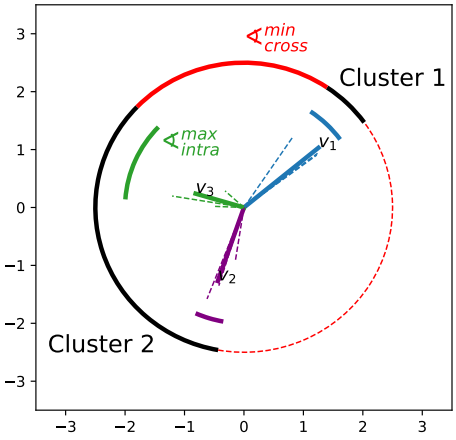


FIGURE B.1: Possible configuration in $d = 2$ with $K = 3$ different data generating distributions and their corresponding gradients v_1, v_2 and v_3. The empirical risk gradients $X_i + v_{i(i)}$, $i = 1, .., M$ are shown as dashed lines. The maximum angles between gradients from the same data generating distribution are shown green, blue and purple in the plot. Among these, the green angle is the largest one $\sphericalangle^{max}_{intra}$. The vectors are optimally bi-partitioned into clusters 1 and 2 and the minimum angle between the gradient updates from any two clients in different clusters $\sphericalangle^{min}_{cross}$ is displayed in red.

Lemma 5. *Let $v, X, Y \in \mathbb{R}^d$ with $\|X\| < \|v\|$ and $\|Y\| < \|v\|$ then*

$$\alpha(v + X, v + Y) \geq -\frac{\|X\|\|Y\|}{\|v\|^2} + \sqrt{1 - \frac{\|X\|^2}{\|v\|^2}}\sqrt{1 - \frac{\|Y\|^2}{\|v\|^2}}. \tag{B.5}$$

Proof. We are interested in vectors X and Y which maximize the angle between $v + X$ and $v + Y$. Since

$$\alpha(v + X, v + Y) = \cos(\sphericalangle(v + X, v + Y)) \tag{B.6}$$

and cos is monotonically decreasing on $[0, \pi]$ such X and Y will minimize the cosine similarity α. As $\|X\| < \|v\|$ and $\|Y\| < \|v\|$ the angle will be maximized if and only if v, X and Y share a common 2-dimensional hyperplane, X is perpendicular to $v + X$ and Y is perpendicular to $v + Y$ and X and Y point into opposite directions (Figure B.2). It then holds by the trigonometric property of the sine that

$$\sin(\sphericalangle(v, v + X)) = \frac{\|X\|}{\|v\|} \tag{B.7}$$

and

$$\sin(\sphericalangle(v, v + Y)) = \frac{\|Y\|}{\|v\|} \tag{B.8}$$

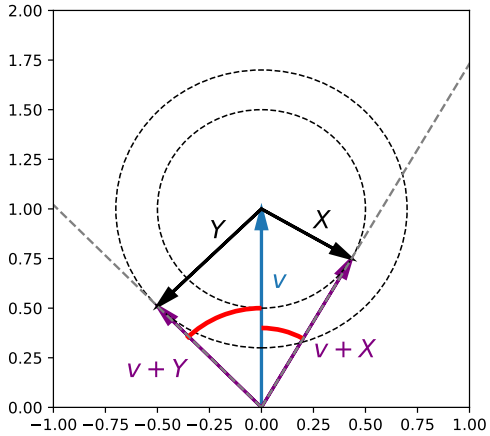


FIGURE B.2: We are interested in a configuration for which the angle between $v + X$ and $v + Y$ is maximized (red in the plot). As $\|X\| < \|v\|$ and $\|Y\| < \|v\|$ this is exactly the case if the line $\{\beta(v + X)|\beta \in \mathbb{R}\}$ is tangential to the circle with center v and radius $\|X\|$ and the line $\{\beta(v + Y)|\beta \in \mathbb{R}\}$ is tangential to the circle with center v and radius $\|Y\|$.

and hence

$$\cos(\sphericalangle(v + X, v + Y)) = \cos(\sphericalangle(v + X) + \sphericalangle(v + Y)) \tag{B.9}$$

$$\geq \cos(\sin^{-1}(\frac{\|X\|}{\|v\|}) + \sin^{-1}(\frac{\|Y\|}{\|v\|})). \tag{B.10}$$

Since

$$\cos(\sin^{-1}(x) + \sin^{-1}(y)) = -xy + \sqrt{1 - x^2}\sqrt{1 - y^2} \tag{B.11}$$

the result follows after re-arranging terms.

□

Lemma 6. *Let $v, w, X, Y \in \mathbb{R}^d$ with $\|X\| < \|v\|$, $\|Y\| < \|w\|$ and define*

$$h(v, w, X, Y) := -\frac{\|X\|\|Y\|}{\|v\|^2} + \sqrt{1 - \frac{\|X\|^2}{\|v\|^2}}\sqrt{1 - \frac{\|Y\|^2}{\|v\|^2}} \tag{B.12}$$

If

$$\frac{\langle v, w\rangle}{\|v\|\|w\|} \leq h(v, w, X, Y) \tag{B.13}$$

then it holds

$$\alpha(v + X, w + Y) \leq \alpha(v, w)h(v, w, X, Y) \tag{B.14}$$

$$+ \sqrt{1 - \alpha(v, w)^2}\sqrt{1 - h(v, w, X, Y)^2} \tag{B.15}$$

Proof. Analogously to the argument in Figure B.2, the angle between $v + X$ and $w + Y$ is minimized, when v, w, X and Y share a common 2-dimensional hyperplane, X is

orthogonal to $v + X$, Y is orthogonal to $w + Y$, and X and Y point towards each other. The minimum possible angle is then given by

$$\begin{aligned}
\sphericalangle(v + X, w + Y) &= \sphericalangle(v, w) - \sphericalangle(v, v + X) - \sphericalangle(w, w + Y) && \text{(B.16)}\\
&\geq \max(0, && \text{(B.17)}\\
&\quad \cos^{-1}(\frac{\langle v, w\rangle}{\|v\|\|w\|}) && \text{(B.18)}\\
&\quad - \sin^{-1}(\frac{\|X\|}{\|v\|}) + && \text{(B.19)}\\
&\quad - \sin^{-1}(\frac{\|Y\|}{\|v\|})) && \text{(B.20)}
\end{aligned}$$

which can be simplified to

$$\begin{aligned}
\sphericalangle(v + X, w + Y) &\geq \max(0, \cos^{-1}(\frac{\langle v, w\rangle}{\|v\|\|w\|}) && \text{(B.21)}\\
&\quad - \cos^{-1}(-\frac{\|X\|\|Y\|}{\|v\|^2} + \sqrt{1 - \frac{\|X\|^2}{\|v\|^2}}\sqrt{1 - \frac{\|Y\|^2}{\|v\|^2}})) && \text{(B.22)}
\end{aligned}$$

Under condition (B.13) then second term in the maximum is greater than zero and we get

$$\begin{aligned}
&\cos(\sphericalangle(v + X, v + Y)) && \text{(B.23)}\\
&\leq \cos(\cos^{-1}(\frac{\langle v, w\rangle}{\|v\|\|w\|}) && \text{(B.24)}\\
&\quad - \cos^{-1}(-\frac{\|X\|\|Y\|}{\|v\|^2} + \sqrt{1 - \frac{\|X\|^2}{\|v\|^2}}\sqrt{1 - \frac{\|Y\|^2}{\|v\|^2}})) && \text{(B.25)}\\
&\leq \cos(\cos^{-1}(\alpha(v, w)) - \cos^{-1}(h(v, w, X, Y))) && \text{(B.26)}
\end{aligned}$$

Since

$$\cos(\cos^{-1}(x) - \cos^{-1}(y)) = xy + \sqrt{1 - x^2}\sqrt{1 - y^2} \tag{B.27}$$

the result follows after re-arranging terms. □

Lemma 7. *Let $v_1, .., v_K \in \mathbb{R}^d$, $d \geq 2$, $\gamma_1, .., \gamma_K \in \mathbb{R}_{>0}$ and*

$$\sum_{i=1}^{K} \gamma_i v_i = 0 \in \mathbb{R}^d \tag{B.28}$$

then there exists a bi-partitioning of the vectors $c_1 \cup c_2 = \{1, .., K\}$ such that

$$\max_{i \in c_1, j \in c_2} \alpha(v_i, v_j) \leq \cos(\frac{\pi}{K - 1}) \tag{B.29}$$

Proof. Lemma 7 can be equivalently stated as follows:

Let $v_1, .., v_K \in \mathbb{R}^d$, $d \geq 2$, $\gamma_1, .., \gamma_K \in \mathbb{R}_{>0}$ and

$$\sum_{i=1}^{K} \gamma_i v_i = 0 \in \mathbb{R}^d \tag{B.30}$$

then there exists a bi-partitioning of the vectors $c_1 \cup c_2 = \{1, .., K\}$ such that

$$\min_{i \in c_1, j \in c_2} \sphericalangle(v_i, v_j) \geq \frac{\pi}{K-1} \tag{B.31}$$

As the angle between two vectors is invariant under multiplication with positive scalars $\gamma > 0$ we can assume w.l.o.g that $\gamma_i = 1$ $i = 1, .., K$.

Let us first consider the case where $d = 2$. Let $e_1 \in \mathbb{R}^2$ be the first standard basis vector and assume w.l.o.g that the vectors $v_1, .., v_K$ are sorted w.r.t. their angular distance to e_1 (they are arranged circular as shows in Figure B.3). As all vectors lie in the 2d plane, we know that the sum of the angles between all neighboring vectors has to be equal to 2π.

$$\sum_{i=1}^{K} \sphericalangle(v_i, v_{(i+1) \bmod K}) = 2\pi \tag{B.32}$$

Now let

$$i_1^* = \arg\max_{i \in \{1,..,K\}} \sphericalangle(v_i, v_{(i+1) \bmod K}) \tag{B.33}$$

and

$$i_2^* = \arg\max_{i \in \{1,..,K\} \setminus i_1^*} \sphericalangle(v_i, v_{(i+1) \bmod K}) \tag{B.34}$$

be the indices of the largest and second largest angles between neighboring vectors and define the following clusters:

$$c_1 = \{i \bmod K | i_1^* < i \leq i_2^* + K[i_2^* < i_1^*]\} \tag{B.35}$$
$$c_2 = \{i \bmod K | i_2^* < i \leq i_1^* + K[i_2^* > i_1^*]\}\} \tag{B.36}$$

where $[x] = 1$ if x is true and $[x] = 0$ is x is false. Then by construction the second largest angle $\sphericalangle(v_{i_2^*}, v_{(i_2^*+1) \bmod K})$ minimizes the angle between any two vectors from the two different clusters c_1, c_2 (see Figure B.3 for an illustration):

$$\min_{i \in c_1, j \in c_2} \sphericalangle(v_i, v_j) = \sphericalangle(v_{i_2^*}, v_{(i_2^*+1) \bmod K}) \tag{B.37}$$

Hence in $d = 2$ we can always find a partitioning c_1, c_2 s.t. the minimum angle between any two vectors from different clusters is greater or equal to the 2nd largest angle between neighboring vectors. This means the worst case configuration of vectors is one where the 2nd largest angle between neighboring vectors is minimized. As the sum of all K angles between neighboring vectors is constant according to (B.32), this is exactly the case when the largest angle between neighboring vectors is maximized and all other $K-1$ angles are equal.

Assume now that the angle between two neighboring vectors is greater than π. That would mean that there exists a separating line l which passes through the origin and all vectors $v_1, .., v_K$ lie on one side of that line. This however is impossible since

$\sum_{l=1}^{K} v_l = 0$. This means that the largest angle between neighboring vectors can not be greater than π. Hence in the worst-case scenario

$$\sphericalangle(v_{i_2^*}, v_{(i_2^*+1) \bmod K}) \geq \frac{2\pi - \pi}{K-1} = \frac{\pi}{K-1}. \tag{B.38}$$

This concludes the proof for $d = 2$.

Now consider he case where $d > 2$. Let c_1, c_2 be a clustering which maximizes the minimum angular distance between any two clients from different clusters. Let

$$i^*, j^* = \arg\min_{i \in c_1, j \in c_2} \sphericalangle(v_i, v_j) \tag{B.39}$$

then v_{i^*} and v_{j^*} are the two vectors with minimal angular distance. Let $A = [v_{i^*}, v_{j^*}] \in \mathbb{R}^{d,2}$ and consider now the projection matrix

$$P = A(A^T A)^{-1} A^T \tag{B.40}$$

which projects all d-dimensional vectors onto the plane spanned by v_{i^*} and v_{j^*}. Then be linearity of the projection we have

$$0 = P0 = P(\sum_{i=1}^{K} v_i) = \sum_{i=1}^{K} P(v_i) \tag{B.41}$$

Hence the projected vectors also satisfy the condition of the Lemma. As

$$\sphericalangle(Pv_{i^*}, Pv_{j^*}) = \sphericalangle(v_{i^*}, v_{j^*}) \tag{B.42}$$

and

$$\sphericalangle(Pv_i, Pv_j) \geq \sphericalangle(v_i, v_j) \tag{B.43}$$

for all $i, j \notin \{i^*, j^*\}$ the clustering c_1, c_2 is still optimal after projecting and we have found a 2d configuration of vectors satisfying the assumptions of Lemma 7 with the same minimal cross-cluster angle. In other words, we have reduced the $d > 2$ case to the $d = 2$ case, for which we have already proven the result. This concludes the proof.

□

Theorem 8 (Separation Theorem)**.** *Let $D_1, .., D_M$ be the local training data of M different clients, each dataset sampled from one of K different data generating distributions $\varphi_1, .., \varphi_K$, such that $D_i \sim \varphi_{I(i)}(x, y)$. Let the empirical risk on every client approximate the true risk at every stationary solution of the Federated Learning objective θ^* s.t.*

$$\|\nabla R_{I(i)}(\theta^*)\| > \|\nabla R_{I(i)}(\theta^*) - \nabla r_i(\theta^*)\| \tag{B.44}$$

and define

$$\gamma_i := \frac{\|\nabla R_{I(i)}(\theta^*) - \nabla r_i(\theta^*)\|}{\|\nabla R_{I(i)}(\theta^*)\|} \in [0, 1) \tag{B.45}$$

Then there exists a bi-partitioning $c_1^* \cup c_2^* = \{1, .., M\}$ of the client population such that that the maximum similarity between the updates from any two clients from different clusters can

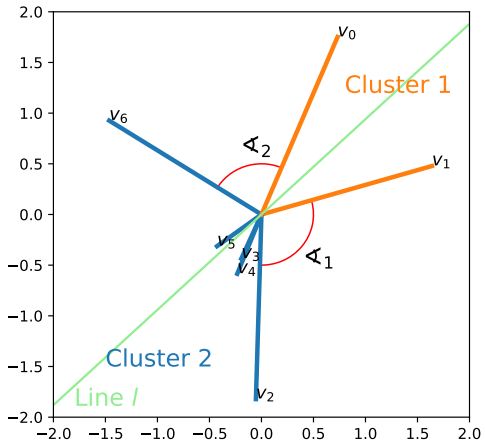


FIGURE B.3: Possible configuration in $d = 2$. The largest and 2nd largest angle between neighboring vectors (red) separate the two optimal clusters. The largest angle between neighboring vectors is never greater than π.

be bounded from above according to

$$\alpha_{cross}^{max} := \min_{c_1 \cup c_2 = \{1,..,M\}} \max_{i \in c_1, j \in c_2} \alpha(\nabla r_i(\theta^*), \nabla r_j(\theta^*)) \tag{B.46}$$

$$= \max_{i \in c_1^*, j \in c_2^*} \alpha(\nabla r_i(\theta^*), \nabla r_j(\theta^*)) \tag{B.47}$$

$$\leq \begin{cases} \cos(\frac{\pi}{K-1}) H_{i,j} + \sin(\frac{\pi}{K-1}) \sqrt{1 - H_{i,j}^2} & \text{if } H \geq \cos(\frac{\pi}{K-1}) \\ 1 & \text{else} \end{cases} \tag{B.48}$$

with

$$H_{i,j} = -\gamma_i \gamma_j + \sqrt{1 - \gamma_i^2} \sqrt{1 - \gamma_j^2} \in (-1, 1]. \tag{B.49}$$

At the same time the similarity between updates from clients which share the same data generating distribution can be bounded from below by

$$\alpha_{intra}^{min} := \min_{\substack{i,j \\ I(i)=I(j)}} \alpha(\nabla_\theta r_i(\theta^*), \nabla_\theta r_j(\theta^*)) \geq \min_{\substack{i,j \\ I(i)=I(j)}} H_{i,j}. \tag{B.50}$$

Proof. For the first result, we know that in every stationary solution of the Federated Learning objective θ^* it holds

$$\sum_{l=1}^{K} \gamma_i \nabla_\theta R_l(\theta^*) = 0 \tag{B.51}$$

and hence by Lemma 7 there exists a bi-partitioning $\hat{c}_1 \cup \hat{c}_2 = \{1, .., K\}$ such that

$$\max_{l \in \hat{c}_1, j \in \hat{c}_2} \alpha(\nabla_\theta R_l(\theta^*), \nabla_\theta R_j(\theta^*)) \leq \cos(\frac{\pi}{K-1}) \tag{B.52}$$

Let

$$c_1 = \{i | I(i) \in \hat{c}_1, i = 1, .., M\} \tag{B.53}$$

and

$$c_2 = \{i | I(i) \in \hat{c}_2, i = 1, .., M\} \tag{B.54}$$

and set for some $i \in c_1$ and $j \in c_2$:

$$v = \nabla_\theta R_{I(i)}(\theta^*) \tag{B.55}$$

$$X = \nabla_\theta r_i(\theta^*) - \nabla_\theta R_{I(i)}(\theta^*) \tag{B.56}$$

$$w = \nabla_\theta R_{I(j)}(\theta^*) \tag{B.57}$$

$$Y = \nabla_\theta r_j(\theta^*) - \nabla_\theta R_{I(j)}(\theta^*) \tag{B.58}$$

Then $\alpha(v, w) \leq \cos(\frac{\pi}{K-1})$ and the result follows directly from Lemma 6.

The second result follows directly from Lemma 5 by setting

$$v = \nabla_\theta R_{I(i)}(\theta^*) \tag{B.59}$$

$$X = \nabla_\theta r_i(\theta^*) - \nabla_\theta R_{I(i)}(\theta^*) \tag{B.60}$$

$$Y = \nabla_\theta r_j(\theta^*) - \nabla_\theta R_{I(i)}(\theta^*) \tag{B.61}$$

□

Appendix C

Federated Learning with Auxiliary Data

C.1 Proof of Theorem 4

The Theorem states that if $R(\cdot)$ is differentiable and 1-strongly convex and l is differentiable with $|l'(z)| \leq 1 \; \forall z$, then the ℓ^2-sensitivity $\Delta_2(\mathcal{M})$ of the mechanism

$$\mathcal{M} : D_i \mapsto \arg\min_w J(w, h_0, D_i, D^-) \tag{C.1}$$

is at most $2(\lambda(|D_i| + |D^-|))^{-1}$.

Proof. The proof is an adaptation of the result shown in (Chaudhuri, Monteleoni, and Sarwate, 2011). We have

$$J(w, h_0, D_i, D^-) = a \sum_{x \in D_i \cup D^-} l(t_x \langle w, \tilde{h}_0(x) \rangle) + \lambda R(w) \tag{C.2}$$

with $t_x = 2(\mathbb{1}_{x \in D_i}) - 1 \in [-1, 1]$, $a = (|D_i| + |D^-|)^{-1}$ and

$$\tilde{h}_0(x) = h_0(x)(\max_{x \in D^- \cup D_i} \|h_0(x)\|)^{-1}. \tag{C.3}$$

Let $D_i = \{x_1, .., x_N\}$ and $D_i' = \{x_1, .., x_N'\}$ be two local data sets that differ in only one element. For arbitrary D^- and h_0 define

$$w^* = \arg\min_w J(w, h_0, D_i, D^-), \tag{C.4}$$

$$v^* = \arg\min_w J(w, h_0, D_i', D^-), \tag{C.5}$$

$$n(w) = J(w, h_0, D_i, D^-) \tag{C.6}$$

and

$$m(w) = J(w, h_0, D_i, D^-) - J(w, h_0, D_i', D^-) \tag{C.7}$$

Since

$$m(w) = a(l(t_x \langle w, h_0(x_N) \rangle) - l(t_x \langle w, h_0(x_N') \rangle)) \tag{C.8}$$

we have

$$\nabla m(w) = a(t_x l'(t_x \langle w, h_0(x_N)\rangle) h_0(x_N)^T - \tag{C.9}$$
$$t_x l'(t_x \langle w, h_0(x'_N)\rangle) h_0(x'_N)^T) \tag{C.10}$$

which can be bounded in norm

$$\|\nabla m(w)\| = a(\|h_0(x_N) - h_0(x'_N)\|) \tag{C.11}$$
$$\leq a(\|h_0(x_N)\| + \|h_0(x'_N)\|) \tag{C.12}$$
$$\leq 2a \tag{C.13}$$

as $t_x \in [-1, 1]$, $|l'(x)| \leq 1$ and

$$\|\tilde{h}_0(x)\| = \|h_0(x)(\max_{x \in D_i \cup D^-} h_0(x))^{-1}\| \leq 1. \tag{C.14}$$

Furthermore, since $n(w)$ is λ-strongly convex it follows by Shalev-Schwartz inequality

$$(\nabla n(w^*) - \nabla n(v^*))^T (w^* - v^*) \geq \lambda \|w^* - v^*\|^2. \tag{C.15}$$

Combining this result with Cauchy-Schwartz inequality and

$$\nabla m(v^*) = \nabla n(v^*) - \nabla n(w^*) \tag{C.16}$$

yields

$$\|w^* - v^*\| \|\nabla m(v^*)\| \geq (w^* - v^*)^T \nabla m(v^*) \tag{C.17}$$
$$= (w^* - v^*)^T (\nabla n(v^*) - \nabla n(w^*)) \tag{C.18}$$
$$\geq \lambda \|w^* - v^*\|^2 \tag{C.19}$$

Thus

$$\|w^* - v^*\| \leq \frac{\|\nabla m(v^*)\|}{\lambda} \leq \frac{2a}{\lambda} \tag{C.20}$$

which concludes the proof. □

C.2 Details on the implementation of different scoring mechanisms

The alternative scoring mechanisms used to obtain the results presented in Figure 7.4 are implemented as follows:

- **Two-class SVM:** To obtain the two-class SVM scoring model, we optimize a Support Vector Machine with RBF kernel in the ERM objective (7.6), instead of a Logistic Regression. We set the kernel bandwidth to $\gamma = 1/(d\sigma^2)$, where d is the dimension of the features extracted by h_0 and σ^2 is the variance in the extracted features $\{h_0(x)|x \in D_i \cup D^-\}$. The scores are then obtained by using a Platt calibration (Platt et al., 1999) with a 5-fold cross-validation.
- **One-class SVM:** We train a one-class Support Vector Machine with RBF kernel on only the extracted features of the local data $\{h_0(x)|x \in D_i\}$ after standard

scaling. Like for the two-class SVM, we set the kernel bandwidth to $\gamma = 1/(d\sigma^2)$. The scores are obtained by transforming the anomaly scores provided by the one-class SVM using a sigmoid function.

- **Isolation Forest:** We train an Isolation Forest F on the extracted features of the local data $\{h_0(x)|x \in D_i\}$ with the number of estimators set to $n = 100$. Scores are obtained by min-max normalizing the anomaly scores (the lower, the more abnormal), which we obtain by evaluating the distillation data $D_{distill}$ using the trained Isolation Forest F.
- **Kernel Density Estimation:** We perform a Gaussian KDE on the extracted features of the local data $\{h_0(x)|x \in D_i\}$. We set the kernel bandwith to $\gamma = 0.5$ and use the evaluation of the computed probability density in the points of the local dataset as scores. All evaluations are then min-max normalized to fall into the range of $[0, 1]$.
- **Random Scoring:** Scores are drawn randomly from a uniform distribution

$$s_i(x) \sim \mathcal{U}([0, 1]). \tag{C.21}$$

- **Equal Scoring:** Scores are assigned as

$$s_i(x) = \frac{1}{2}. \tag{C.22}$$

C.3 Additional Results and Detailed Training Curves

In this sections we give detailed training curves for the results shown in Figure 7.5. As can be seen, in the highly non-iid setting at $\alpha \in \{0.01, 0.04\}$, all methods exhibit convergence issues. This behavior is well known in FL and is described for instance in (Zhao et al., 2018; Sattler et al., 2020b). Notably, the performance of FEDAUX after one single communication round exceeds the maximum achieved performance of all other methods over the entire course of training. At higher values of $\alpha \geq 0.16$ all methods train smoothly and validation performance asymptotically increases over the curse of training. FEDAUX dominates all baseline methods at all communication rounds in the heterogeneous settings. In the mostly iid-setting at $\alpha = 10.24$ FEDAUX is en par with the pre-trained version of FEDDF.

Table C.1 compares performance of FEDAUX to baseline methods on the CIFAR-100 data set. Again FEDAUX outperforms FEDAVG and FEDDF across all level of data heterogeneity α and shows superior performance to the improved FEDDF+P when data is highly heterogeneous at $\alpha = \{0.01, 0.04\}$. Interestingly in this setting FEDDF+P manages to slightly outperform FEDAUX at medium data heterogeneity levels $\alpha = \{0.16, 0.64\}$. This indicates that our proposed differentially private certainty scoring method may insufficiently approximate the true client certainty in this setting. We leave potential improvements of this mechanism for future work.

C.4 Details on generating Imagenet subsets

To simulate the effects of a wide variety of auxiliary data sets on the training performance of FEDAUX, we generate different structured subsets of the ImageNet data base (resized to $32 \times 32 \times 3$). Each subset is defined via a top-level Wordnet ID which

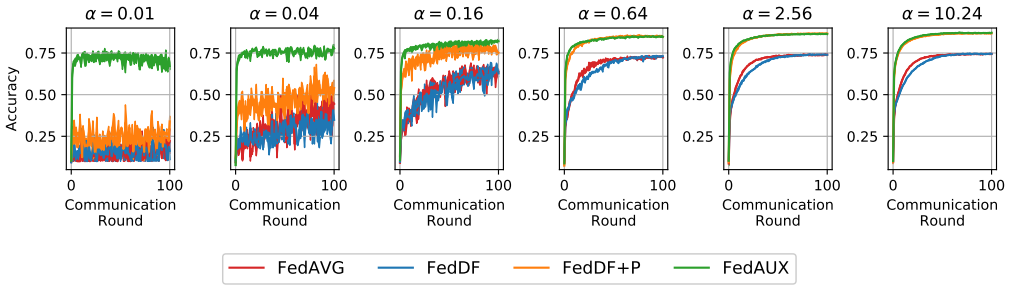


FIGURE C.1: Detailed training curves for ResNet-8 trained on CIFAR-10, $n = 80$ Clients, $C = 40\%$.

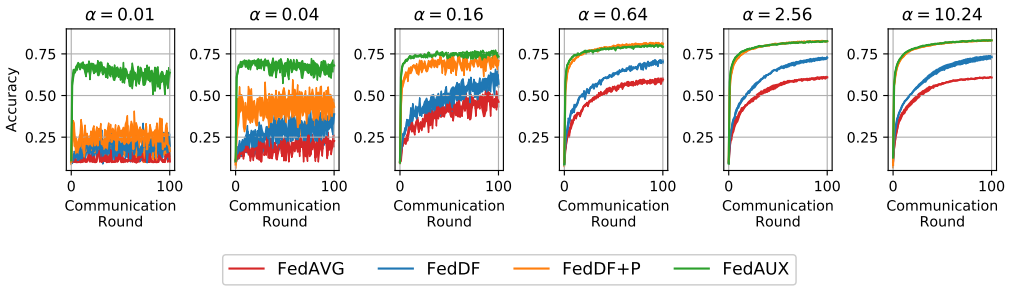


FIGURE C.2: Detailed training curves for MobileNetv2 trained on CIFAR-10, $n = 100$ Clients, $C = 40\%$.

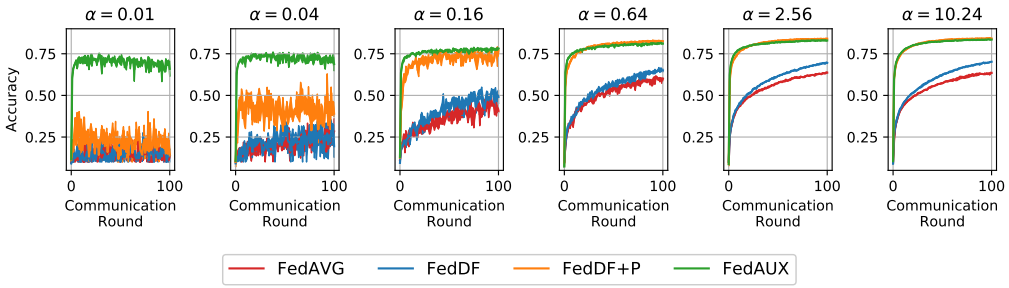


FIGURE C.3: Shufflenet trained on CIFAR-10, $n = 100$ Clients, $C = 40\%$.

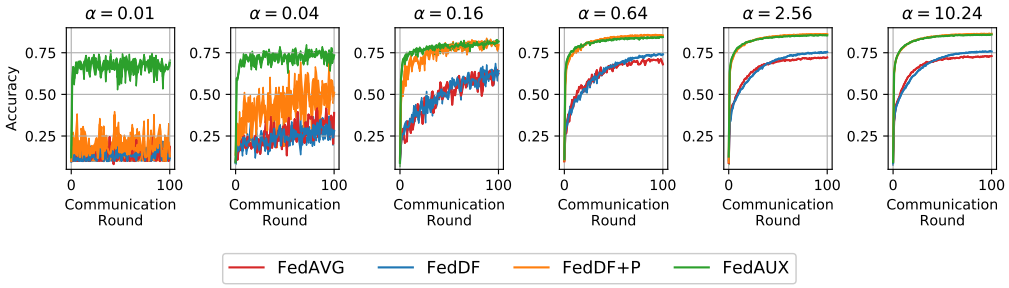


FIGURE C.4: Detailed training curves for mixed models trained on CIFAR-10. 20 each train ResNet8, MobileNetv2 and Shufflenet respectively.

is shown in Table C.2. To obtain the images from the subset, we select all leaf-node IDs of the respective top-level IDs via the Imagenet API

`http://www.image-net.org/api/text/wordnet.structure.hyponym?wnid=<top-levelID>&full=1`

and then take only those classes from the full Imagenet data set, which match these leaf-node IDs. Table C.2 also shows the number of samples contained in every subset

TABLE C.1: Results on data sets with higher number of classes. Training ResNet-8 on CIFAR-100. Accuracy achieved after $T = 100$ communication rounds by different Federated Distillation methods at different levels of data heterogeneity α. STL-10 is used as auxiliary data set.

	α					
	0.01	0.04	0.16	0.64	2.56	10.24
FedAVG	24.1	36.3	47.2	50.7	52.2	52.2
FedDF	11.4	24.4	45.0	49.5	52.5	51.2
FedDF+P	18.2	42.0	58.0	60.8	61.6	62.0
FedAUX	34.1	47.4	56.4	60.7	62.5	62.5

TABLE C.2: Auxiliary data sets used in this study and their defining Wordnet IDs and data sets sizes.

Data set	Wordnet ID	Dataset Size
Imagenet Devices	n03183080	165747
Imagenet Birds	n01503061	76541
Imagenet Animals	n00015388	510530
Imagenet Dogs	n02084071	147873
Imagenet Invertebrates	n01905661	79300
Imagenet Structures	n04341686	74400

that was generated this way.

C.5 Details on the Implementation and Results of the NLP Benchmarks

As mentioned in section 4.3 *Evaluating* FEDAUX *on NLP Benchmarks* we used TinyBERT as a model for our NLP experiments. TinyBERT was pre-trained on Bookcorpus[1] which led us to select the same data set as a public data set in order to follow the methodology outlined in section 7.1.3. As private data sets we chose the AG News data set[2] (Zhang, Zhao, and LeCun, 2015b), a topic classification data set, and the english texts from the Multilingual Amazon Reviews Corpus[3] (Keung et al., 2020), which we use for predicting how many stars a review gets. The pre-trained weights and the tokenizer for TinyBERT are available at the corresponding repository[4]. All experiments were conducted using $\epsilon = 0.1$ and $\delta = 10^{-5}$ as differential privacy parameters, 1 epoch for local training and distillation, ten clients and 100% participation rate as well as 160000 disjoint data points, which were sampled from BookCorpus, for the public and distillation data sets respectively. Furthermore the ADAM optimizer with a learning rate of 10^{-5} was used for both local training and distillation. The regularization strength of the logistic regression classifier was set to 0.01. The batch size for D_i, D^- and $D_{distill}$ was 32. Detailed results for figure 7.6 are depicted in table C.3.

[1] https://huggingface.co/datasets/bookcorpus
[2] https://huggingface.co/datasets/ag_news
[3] https://huggingface.co/datasets/amazon_reviews_multi
[4] https://huggingface.co/huawei-noah/TinyBERT_General_4L_312D

TABLE C.3: NLP Benchmarks of different FL methods. Maximum accuracy achieved after $T = 20$ communication rounds at participation-rate $C = 100\%$.

	AG News		Amazon	
Method	$\alpha = 0.01$	$\alpha = 1.0$	$\alpha = 0.01$	$\alpha = 1.0$
FEDAVG+P	78.80±4.40	92.17±1.98	41.70±0.58	55.17±0.40
FEDDF+P	78.05±7.64	90.83±0.25	38.04±0.84	54.63±0.66
FEDAUX	85.04±1.21	91.00±0.30	49.11±0.22	54.86±0.61

TABLE C.4: Best performing hyperparameter combinations for each method when training ResNet8 with $n = 20$ clients for 50 communication rounds at a participation rate of $C = 40\%$. Methods sorted by top accuracy.

Method	Alpha	Local LR	Distill LR	λ FedProx	Accuracy
FedPROX+P	100	0.001	-	0.0001	0.8946
FedAUX		0.001	1e-05	-	0.8941
FedDF+P		0.001	1e-05	-	0.8936
FedAVG+P		0.001	-	-	0.8924
FedBE		0.001	1e-05	-	0.8246
FedPROX		0.001	-	0.001	0.8232
FedAVG		0.001	-	-	0.8228
FedDF		0.001	1e-05	-	0.8210
FedAUX	0.01	0.001	0.0001	-	0.7501
FedPROX+P		0.01	-	0.01	0.6122
FedDF+P		0.001	0.001	-	0.4786
FedPROX		0.001	-	0.01	0.4145
FedAVG+P		0.001	-	-	0.3929
FedDF		0.001	0.001	-	0.3481
FedBE		0.001	0.001	-	0.3196
FedAVG		0.0001	-	-	0.2770

C.6 Hyperparameter Evaluation

In this section we provide a detailed hyperparameter analysis for our proposed method and the baseline methods used in this study. For all methods we use the very popular Adam optimizer for both local training and distillation. We vary the learning rate in $\{1e-2, 1e-3, 1e-4, 1e-5\}$ for local training an distillation. For FedPROX, we vary the parameter λ_{prox}, controlling the proximal term in the training objective in $\{1e-2, 1e-3, 1e-4, 1e-5\}$. Figure C.5 compares the maximum achieved accuracy after 50 communication rounds for the different methods and hyperparameter settings, for a FL setting with 20 clients training ResNet-8 on CIFAR-10 at a participation-rate of 40%. The auxiliary data set we use is STL-10.

For each method and each level of data heterogeneity, table C.4 shows the accuracy of the best performing combination of hyperparameters. As we can see FEDAUX matches the performance of the best performing methods in the iid setting with $\alpha = 100.0$ and outperforms all other methods distinctively in the non-iid setting with $\alpha = 0.01$.

C.7 Empirical Privacy Evaluation

Our proposed method is provably differentially private and achieves state-of-the-art performance, even at very conservative privacy levels. If not explicitly stated otherwise, all results presented in this study were achieved with (ε, δ)-differentially

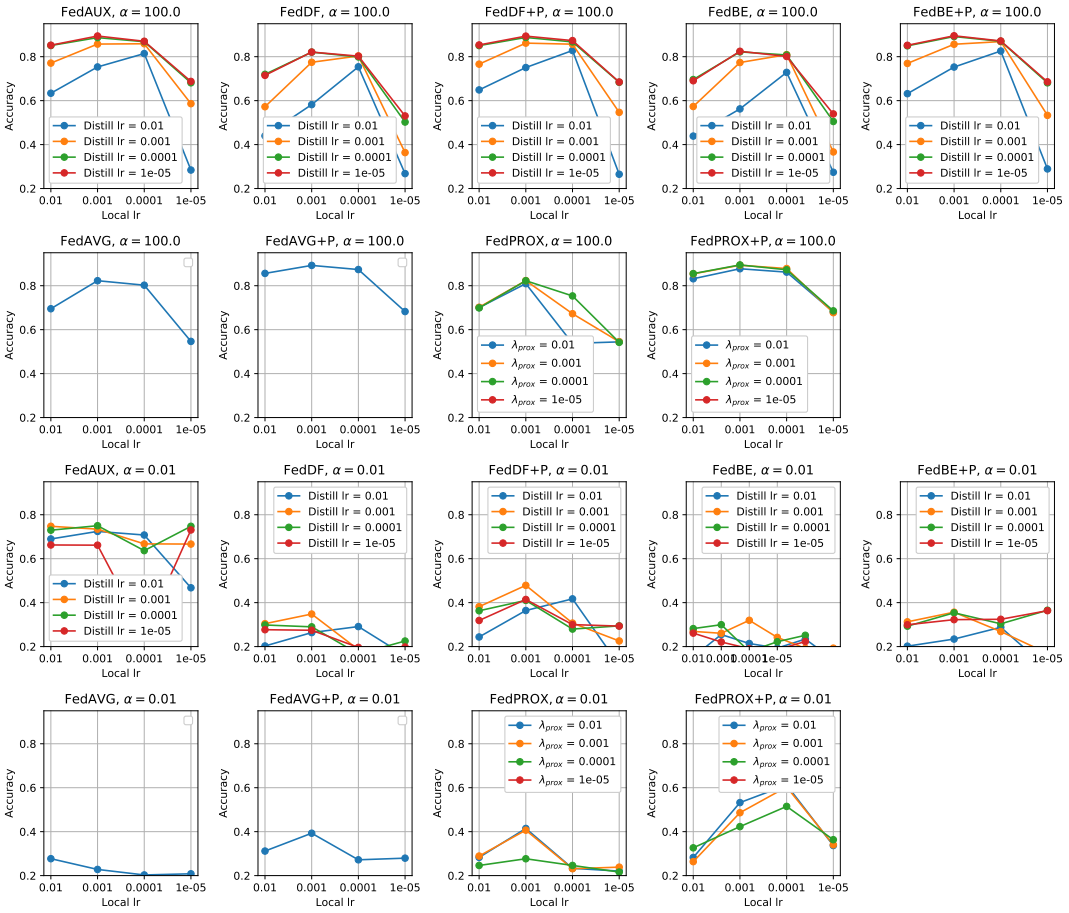
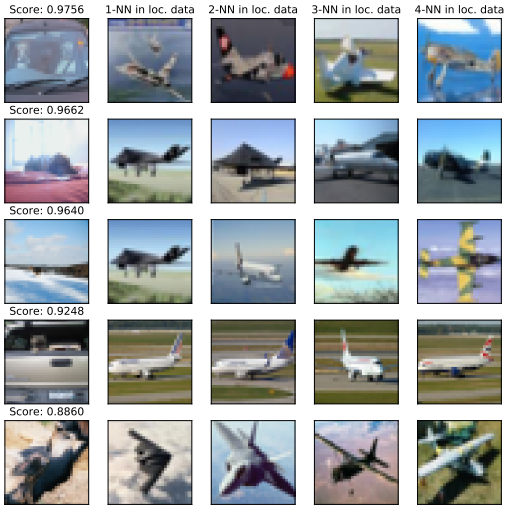


FIGURE C.5: Results of our hyperparameter optimization for ResNet8. 20 Clients are trained for 50 communication rounds, at a participation rate of $C = 40\%$. Both local training and distillation is performed for 1 epoch.

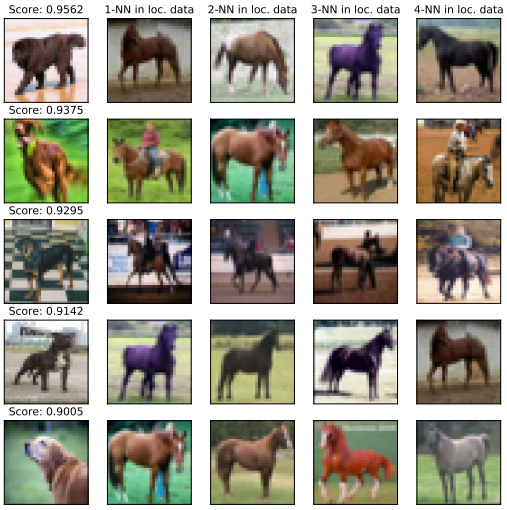
private certainty scores at conservative privacy parameters $\delta = 10^{-5}$ and $\varepsilon = 0.1$. In this section, we additionally evaluate the privacy properties of the certainty scores empirically. Figure C.6 shows, for four different clients, the 5 images x from the distillation data set $D_{distill}$, which were assigned the highest certainty score $s_i(x)$ by the client's scoring model w_i^* (left column). Displayed next to the images are their 4 nearest neighbors x' in feature space which maximize the cosine-similarity

$$\text{sim}(x, x') = \frac{\langle h_0(x), h_0(x') \rangle}{\|h_0(x)\| \|h_0(x')\|}. \tag{C.23}$$

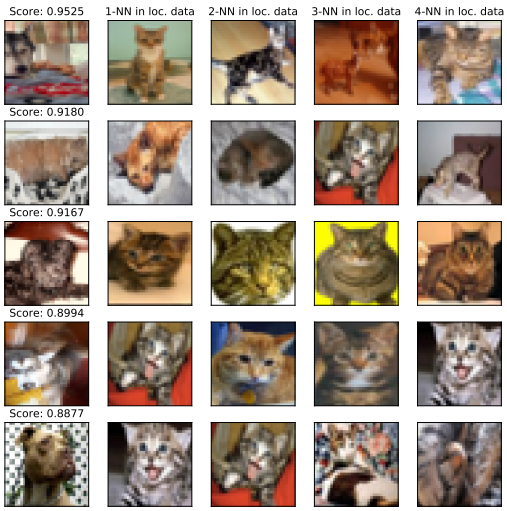
In this example the clients hold non-iid subsets of CIFAR-10 ($\alpha = 0.01$) and the "Imagenet Dogs" (c.f. Appendix C.4) data set is used as auxiliary data. Using weighted ensemble distillation in this setting improves training performance from 48.46% to 75.59%. As we can see, while certainty scores are able to inform the distillation process and allow FEDAUX to outperform baseline methods on heterogeneous data, they reveal only fuzzy, indirect information about the local training data. For instance, client 1, which in this example is mainly holding data from the airplane class, assigns the highest scores to pictures in the auxiliary data set that show dogs in cars or in front of blue skies. From this it could be concluded that a majority of the clients training data contains man-made objects in front of blue backgrounds, but direct exposure of single data points is improbable.



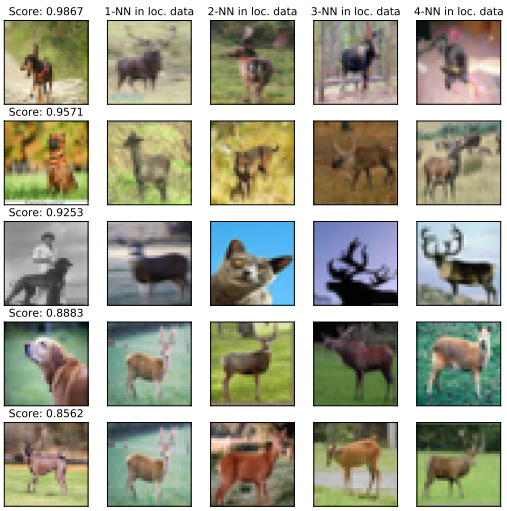
(a) Client 1: Images x from the distill data set with the highest scores $s_i(x)$ and their nearest neighbors in feature space in the local data set D_i.



(b) Client 2: Images x from the distill data set with the highest scores $s_i(x)$ and their nearest neighbors in feature space in the local data set D_i.



(c) Client 3: Images x from the distill data set with the highest scores $s_i(x)$ and their nearest neighbors in feature space in the local data set D_i.



(d) Client 4: Images x from the distill data set with the highest scores $s_i(x)$ and their nearest neighbors in feature space in the local data set D_i.

FIGURE C.6: Data points x from the auxiliary data set which were assigned the highest scores $s_i(x)$ and their nearest neighbors in the data of 4 randomly selected clients D_i. Clients hold non-iid subsets from the CIFAR-10 data set ($\alpha = 0.01$). Auxiliary data used is ImageNet Dogs (cf. Appendix C.4). No differential privacy is used.

Note that there exist also many FL scenarios in which the server is assumed to be trustworthy, and only the final trained model which is released to the public needs to be privately sanitized. In these settings, direct inspection of certainty scores by outside adversaries is not possible and thus privacy loss through certainty scores is even less critical. Future work could also explore the use encryption-based techniques for secure weighted aggregation of client predictions.

Bibliography

Abadi, Martin et al. (2016). "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 308–318.

Abdar, Moloud et al. (2021). "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". In: *Information Fusion*.

Aji, Alham Fikri and Kenneth Heafield (2017). "Sparse Communication for Distributed Gradient Descent". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Association for Computational Linguistics, pp. 440–445. DOI: 10.18653/v1/d17-1045. URL: https://doi.org/10.18653/v1/d17-1045.

Alistarh, Dan et al. (2017). "QSGD: Communication-efficient SGD via gradient quantization and encoding". In: *Advances in Neural Information Processing Systems*, pp. 1709–1720.

Amiri, Mohammad Mohammadi and Deniz Gündüz (2020a). "Federated learning over wireless fading channels". In: *IEEE Transactions on Wireless Communications* 19.5, pp. 3546–3557.

— (2020b). "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air". In: *IEEE Transactions on Signal Processing* 68, pp. 2155–2169.

Bagdasaryan, Eugene et al. (2020). "How To Backdoor Federated Learning". In: *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*. Vol. 108. Proceedings of Machine Learning Research. PMLR, pp. 2938–2948. URL: http://proceedings.mlr.press/v108/bagdasaryan20a.html.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. URL: http://arxiv.org/abs/1409.0473.

Bellet, Aurélien et al. (2018). "Personalized and Private Peer-to-Peer Machine Learning". In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. Vol. 84. Proceedings of Machine Learning Research. PMLR, pp. 473–481. URL: http://proceedings.mlr.press/v84/bellet18a.html.

Bernstein, Jeremy et al. (2018). "SIGNSGD: Compressed Optimisation for Non-Convex Problems". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 559–568. URL: http://proceedings.mlr.press/v80/bernstein18a.html.

Bhagoji, Arjun Nitin et al. (2019). "Analyzing Federated Learning through an Adversarial Lens". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 634–643. URL: http://proceedings.mlr.press/v97/bhagoji19a.html.

Bhowmick, Abhishek et al. (2018). "Protection against reconstruction and its applications in private federated learning". In: *arXiv preprint arXiv:1812.00984*.

Bistritz, Ilai, Ariana J. Mann, and Nicholas Bambos (2020). "Distributed Distillation for On-Device Learning". In: *34th Conference on Neural Information Processing Systems (NeurIPS)*.

Blanchard, Peva, Rachid Guerraoui, Julien Stainer, et al. (2017). "Machine learning with adversaries: Byzantine tolerant gradient descent". In: *Advances in Neural Information Processing Systems*, pp. 119–129.

Bonawitz, Keith et al. (2017). "Practical secure aggregation for privacy-preserving machine learning". In: *2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191.

Bonawitz, Keith et al. (2019). "Towards Federated Learning at Scale: System Design". In: *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org. URL: https://proceedings.mlsys.org/book/271.pdf.

Bosse, Sebastian et al. (2018). "Deep Neural Networks for No-Reference and Full-Reference Image Quality Assessment". In: *IEEE Transactions on Image Processing* 27.1, pp. 206–219.

Briggs, Christopher, Zhong Fan, and Peter Andras (2020). "Federated learning with hierarchical clustering of local updates to improve training on non-IID data". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–9.

Brown, Tom B. et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33.

Caldas, Sebastian et al. (2018a). "Expanding the Reach of Federated Learning by Reducing Client Resource Requirements". In: *CoRR* abs/1812.07210. arXiv: 1812.07210. URL: http://arxiv.org/abs/1812.07210.

Caldas, Sebastian et al. (2018b). "Leaf: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097*.

Carlini, Nicholas et al. (2018). "The secret sharer: Measuring unintended neural network memorization & extracting secrets". In: *arXiv preprint arXiv:1802.08232*.

Caruana, Rich (1997). "Multitask learning". In: *Machine learning* 28.1, pp. 41–75.

Chang, Hongyan et al. (2019). "Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer". In: *arXiv preprint arXiv:1912.11279*.

Chaudhuri, Kamalika, Claire Monteleoni, and Anand D. Sarwate (2011). "Differentially Private Empirical Risk Minimization". In: *J. Mach. Learn. Res.* 12, pp. 1069–1109.

Chen, Bryant et al. (2018a). "Detecting backdoor attacks on deep neural networks by activation clustering". In: *arXiv preprint arXiv:1811.03728*.

Chen, Hong-You and Wei-Lun Chao (2020). "FedDistill: Making bayesian model ensemble applicable to federated learning". In: *arXiv preprint arXiv:2009.01974*.

Chen, Jianmin et al. (2016a). "Revisiting distributed synchronous SGD". In: *arXiv preprint arXiv:1604.00981*.

Chen, Lingjiao et al. (2018b). "DRACO: Byzantine-resilient Distributed Training via Redundant Gradients". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 902–911. URL: http://proceedings.mlr.press/v80/chen18l.html.

Chen, Mingzhe et al. (2020a). "Convergence Time Minimization of Federated Learning over Wireless Networks". In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148815.

Chen, Ting et al. (2020b). "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 1597–1607.

Chen, Xuhui et al. (2018c). "When machine learning meets blockchain: A decentralized, privacy-preserving and secure design". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1178–1187.

Chen, Yudong, Lili Su, and Jiaming Xu (2017). "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.2, p. 44.

Chen, Yunji et al. (2016b). "DianNao family: energy-efficient hardware accelerators for machine learning". In: *Communications of the ACM* 59.11, pp. 105–112.

Chilimbi, Trishul M et al. (2014). "Project Adam: Building an Efficient and Scalable Deep Learning Training System." In: *OSDI*. Vol. 14, pp. 571–582.

Chong, Kai Fong Ernest (2020). "A closer look at the approximation capabilities of neural networks". In: *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. URL: https://openreview.net/forum?id=rkevSgrtPr.

Coates, Adam, Andrew Ng, and Honglak Lee (2011). "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 215–223.

Cohen, Gregory et al. (2017). "EMNIST: an extension of MNIST to handwritten letters". In: *arXiv preprint arXiv:1702.05373*.

Corinzia, Luca and Joachim M Buhmann (2019). "Variational Federated Multi-Task Learning". In: *arXiv preprint arXiv:1906.06268*.

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David (2015). "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28, pp. 3123–3131.

Courtiol, Pierre et al. (2019). "Deep learning-based classification of mesothelioma improves prediction of patient outcome". In: *Nature medicine* 25.10, pp. 1519–1525.

Creemers, R (2016). "Cybersecurity Law of the People's Republic of China (Third Reading Draft)". In: *China Copyright and Media*.

De Sa, Christopher et al. (2015). "Taming the wild: A unified analysis of hogwild!-style algorithms". In: *Advances in neural information processing systems* 28, p. 2656.

Dean, Jeffrey and Sanjay Ghemawat (2008). "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1, pp. 107–113.

Deng, Jia et al. (2009). "ImageNet: A large-scale hierarchical image database". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL-HLT)*. Vol. 1, pp. 4171–4186.

Ding, Zhiguo et al. (2017). "Application of Non-Orthogonal Multiple Access in LTE and 5G Networks". In: *IEEE Communications Magazine* 55.2, pp. 185–191. DOI: 10.1109/MCOM.2017.1500657CM.

Duan, Moming et al. (2020). "FedGroup: Ternary Cosine Similarity-based Clustered Federated Learning Framework toward High Accuracy in Heterogeneity Data". In: *arXiv preprint arXiv:2010.06870*.

Dwork, Cynthia and Aaron Roth (2014). "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3-4, pp. 211–407.

Fang, Minghong et al. (2020). "Local model poisoning attacks to Byzantine-robust federated learning". In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 1605–1622.

Fredrikson, Matt, Somesh Jha, and Thomas Ristenpart (2015). "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 1322–1333.

Fung, Clement, Chris JM Yoon, and Ivan Beschastnikh (2018). "Mitigating sybils in federated learning poisoning". In: *arXiv preprint arXiv:1808.04866*.

Geiping, Jonas et al. (2020). "Inverting Gradients - How easy is it to break privacy in federated learning?" In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL: https://proceedings.neurips.cc/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html.

Geyer, Robin C, Tassilo Klein, and Moin Nabi (2017). "Differentially private federated learning: A client level perspective". In: *arXiv preprint arXiv:1712.07557*.

Ghosh, Avishek et al. (2019). "Robust Federated Learning in a Heterogeneous Environment". In: *arXiv preprint arXiv:1906.06629*.

Goldreich, Oded (1998). "Secure multi-party computation". In: *Manuscript. Preliminary version* 78.

Golomb, Solomon (1966). "Run-length encodings (Corresp.)" In: *IEEE transactions on information theory* 12.3, pp. 399–401.

Goodfellow, Ian J et al. (2013). "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211*.

Graves, Alex and Jürgen Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5-6, pp. 602–610.

Guha, Neel, Ameet Talwalkar, and Virginia Smith (2019). "One-Shot Federated Learning". In: *arXiv preprint arXiv:1902.11175*.

Hard, Andrew et al. (2018). "Federated Learning for Mobile Keyboard Prediction". In: URL: https://arxiv.org/abs/1811.03604.

Hashem, Sherif and Bruce Schmeiser (1993). "Approximating a function and its derivatives using MSE-optimal linear combinations of trained feedforward neural networks". In: *Proceedings of the World Congress on Neural Networks*. Vol. 1, pp. 617–620.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hinton, Geoffrey, Oriol Vinyals, and Jeffrey Dean (2015). "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop*. URL: http://arxiv.org/abs/1503.02531.

Hinton, Geoffrey et al. (2012). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97.

Hitaj, Briland, Giuseppe Ateniese, and Fernando Perez-Cruz (2017). "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, pp. 603–618.

Hoffman, Judy, Mehryar Mohri, and Ningshan Zhang (2018). "Algorithms and Theory for Multiple-Source Adaptation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31, pp. 8256–8266.

Hsu, Tzu-Ming Harry, Hang Qi, and Matthew Brown (2019). "Measuring the effects of non-identical data distribution for federated visual classification". In: *arXiv preprint arXiv:1909.06335*.

Huang, Gao et al. (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2, p. 3.

Huang, Yanping et al. (2019). "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32, pp. 103–112.

Hwang, Kai (2017). *Cloud computing for machine learning and cognitive applications*. MIT Press.

Inan, Hakan, Khashayar Khosravi, and Richard Socher (2017). "Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=r1aPbsFle.

Ioffe, Sergey (2017). "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models". In: *Advances in Neural Information Processing Systems*, pp. 1945–1953.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*, pp. 448–456.

Itahara, Sohei et al. (2020). "Distillation-Based Semi-Supervised Federated Learning for Communication-Efficient Collaborative Training with Non-IID Private Data". In: *arXiv preprint arXiv:2008.06180*.

Jacob, Laurent, Jean-philippe Vert, and Francis R Bach (2009). "Clustered multi-task learning: A convex formulation". In: *Advances in neural information processing systems*, pp. 745–752.

Jeong, Eunjeong et al. (2018). "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data". In: *arXiv preprint arXiv:1811.11479*.

Jeong, Wonyong et al. (2020). "Federated semi-supervised learning with inter-client consistency". In: *arXiv preprint arXiv:2006.12097*.

Jia, Xianyan et al. (2018). "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes". In: *arXiv preprint arXiv:1807.11205*.

Jiang, Peng and Gagan Agrawal (2018). "A linear speedup analysis of distributed deep learning with sparse and quantized communication". In: *Advances in Neural Information Processing Systems*, pp. 2525–2536.

Jiao, Xiaoqi et al. (2020). "TinyBERT: Distilling BERT for Natural Language Understanding". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP)*, pp. 4163–4174.

Jiménez, Daniel (1998). "Dynamically weighted ensemble neural networks for classification". In: *IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence*. Vol. 1, pp. 753–756.

Kairouz, Peter et al. (2019). "Advances and open problems in federated learning". In: *arXiv preprint arXiv:1912.04977*.

Karpathy, Andrej and Li Fei-Fei (2015). "Deep visual-semantic alignments for generating image descriptions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137.

Karpathy, Andrej et al. (2014). "Large-scale video classification with convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732.

Keung, Phillip et al. (2020). "The Multilingual Amazon Reviews Corpus". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4563–4568.

Kidger, Patrick and Terry J. Lyons (2020). "Universal Approximation with Deep Narrow Networks". In: *Conference on Learning Theory (COLT)*. Vol. 125. Proceedings of Machine Learning Research, pp. 2306–2327.

Kim, Yoon et al. (2016). "Character-Aware Neural Language Models". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, pp. 2741–2749. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12489.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. URL: http://arxiv.org/abs/1412.6980.

Koloskova, Anastasia et al. (2020). "Decentralized Deep Learning with Arbitrary Communication Compression". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. URL: https://openreview.net/forum?id=SkgGCkrKvH.

Konecný, Jakub et al. (2016). "Federated Learning: Strategies for Improving Communication Efficiency". In: *CoRR* abs/1610.05492. arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492.

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2014). "The CIFAR-10 dataset". In: *online: http://www. cs. toronto. edu/kriz/cifar. html*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Kumar, Abhishek and Hal Daumé III (2012). "Learning Task Grouping and Overlap in Multi-task Learning". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress. URL: http://icml.cc/2012/papers/690.pdf.

Lalitha, Anusha et al. (2019). "Peer-to-peer federated learning on graphs". In: *arXiv preprint arXiv:1901.11173*.

LeCun, Yann (1998). "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann, John S. Denker, and Sara A. Solla (1990). "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 2, pp. 598–605.

LeCun, Yann et al. (1989). "Handwritten digit recognition with a back-propagation network". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 2, pp. 396–404.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann A et al. (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Lewis, Theodore Gyle (1994). *Foundations of parallel programming - a machine-independent approach*. IEEE. ISBN: 978-0-8186-5692-7.

Li, Daliang and Junpu Wang (2019). "FedMD: Heterogenous federated learning via model distillation". In: *arXiv preprint arXiv:1910.03581*.

Li, Fengfu, Bo Zhang, and Bin Liu (2016). "Ternary Weight Networks". In: *arXiv preprint arXiv:1605.04711*.

Li, Qinbin, Zeyi Wen, and Bingsheng He (2019). "Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection". In: *CoRR* abs/1907.09693. arXiv: 1907.09693. URL: http://arxiv.org/abs/1907.09693.

Li, Tian et al. (2019). "Privacy for Free: Communication-Efficient Learning with Differential Privacy Using Sketches". In: *CoRR* abs/1911.00972. arXiv: 1911.00972. URL: http://arxiv.org/abs/1911.00972.

Li, Tian et al. (2020a). "Federated Optimization in Heterogeneous Networks". In: *Proceedings of Machine Learning and Systems (MLSys)*.

Li, Xiang et al. (2020b). "On the convergence of FedAvg on non-iid data". In: *Proceedings of 8th International Conference on Learning Representations (ICLR)*. OpenReview.net.

Lin, Tao et al. (2020a). "Don't Use Large Mini-batches, Use Local SGD". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. URL: https://openreview.net/forum?id=B1eyO1BFPr.

Lin, Tao et al. (2020b). "Ensemble Distillation for Robust Model Fusion in Federated Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL: https://proceedings.neurips.cc/paper/2020/hash/18df51b97ccd68128e994804f3eccc87-Abstract.html.

Lin, Yujun et al. (2018). "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SkhQHMW0W.

Liu, Dong C. and Jorge Nocedal (1989). "On the limited memory BFGS method for large scale optimization". In: *Math. Program.* 45.1-3, pp. 503–528.

Mansour, Yishay, Mehryar Mohri, and Afshin Rostamizadeh (2008). "Domain Adaptation with Multiple Sources". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 21, pp. 1041–1048.

Marcus, Mitchell P, Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2, pp. 313–330.

Marpe, Detlev, Heiko Schwarz, and Thomas Wiegand (2003). "Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard". In: *IEEE Trans. Circuits Syst. Video Technol.* 13.7, pp. 620–636.

Masoudnia, Saeed and Reza Ebrahimpour (2014). "Mixture of experts: A literature survey". In: *Artif. Intell. Rev.* 42.2, pp. 275–293.

McLeod, Alexander and Diane Dolezel (2018). "Cyber-analytics: Modeling factors associated with healthcare data breaches". In: *Decision Support Systems* 108, pp. 57–68.

McMahan, Brendan et al. (2017). "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282.

Melis, Luca et al. (2019). "Exploiting Unintended Feature Leakage in Collaborative Learning". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, pp. 691–706. DOI: 10.1109/SP.2019.00029. URL: https://doi.org/10.1109/SP.2019.00029.

Merity, Stephen et al. (2017). "Pointer Sentinel Mixture Models". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017,*

Conference Track Proceedings. OpenReview.net. URL: https://openreview.net/forum?id=Byj72udxe.

Mhamdi, El Mahdi El, Rachid Guerraoui, and Sébastien Rouault (2018). "The Hidden Vulnerability of Distributed Learning in Byzantium". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 3518–3527. URL: http://proceedings.mlr.press/v80/mhamdi18a.html.

Mills, Jed, Jia Hu, and Geyong Min (2019). "Communication-efficient federated learning for wireless edge intelligence in IoT". In: *IEEE Internet of Things Journal* 7.7, pp. 5986–5994.

Mohassel, Payman and Yupeng Zhang (2017). "Secureml: A system for scalable privacy-preserving machine learning". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 19–38.

Montavon, Grégoire, Wojciech Samek, and Klaus-Robert Müller (2018). "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* 73, pp. 1–15. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2017.10.011. URL: https://www.sciencedirect.com/science/article/pii/S1051200417302385.

Moritz, Philipp et al. (2016). "SparkNet: Training Deep Networks in Spark". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. URL: http://arxiv.org/abs/1511.06051.

Muñoz-González, Luis, Kenneth T Co, and Emil C Lupu (2019). "Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging". In: *arXiv preprint arXiv:1909.05125*.

Muñoz-González, Luis et al. (2017). "Towards poisoning of deep learning algorithms with back-gradient optimization". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, pp. 27–38.

Nayak, Gaurav Kumar et al. (2019). "Zero-Shot Knowledge Distillation in Deep Networks". In: *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, pp. 4743–4751.

Nelus, Alexandru, Rene Glitza, and Rainer Martin (2021). "Estimation of Microphone Clusters in Acoustic Sensor Networks using Unsupervised Federated Learning". In: *arXiv preprint arXiv:2102.03109*.

Neumann, David et al. (2019). "DeepCABAC: Plug&Play Compression of Neural Network Weights and Weight Updates". In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 21–25. DOI: 10.1109/ICIP40778.2020.9190821. URL: https://dx.doi.org/10.1109/ICIP40778.2020.9190821.

Oala, Luis et al. (2020). "Interval Neural Networks: Uncertainty Scores". In: *arXiv preprint arXiv:2003.11566*.

Opitz, David W. and Richard Maclin (1999). "Popular Ensemble Methods: An Empirical Study". In: *J. Artif. Intell. Res.* 11, pp. 169–198.

Papernot, Nicolas et al. (2018). "Scalable Private Learning with PATE". In: *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. OpenReview.net.

Perrone, Michael P. and Leon N Cooper (1993). "When networks disagree: Ensemble methods for hybrid neural networks". In: *Neural Networks for Speech and Image Processing*. Chapman and Hall.

Platt, John et al. (1999). "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". In: *Advances in large margin classifiers* 10.3, pp. 61–74.

Qayyum, Adnan et al. (2021). "Collaborative Federated Learning For Healthcare: Multi-Modal COVID-19 Diagnosis at the Edge". In: *arXiv preprint arXiv:2101.07511*.

Radford, Alec et al. (2019). "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8, p. 9.

Ren, Jinke, Guanding Yu, and Guangyao Ding (2020). "Accelerating DNN training in wireless federated edge learning systems". In: *IEEE Journal on Selected Areas in Communications* 39.1, pp. 219–232.

Ruder, Sebastian (2017). "An overview of multi-task learning in deep neural networks". In: *arXiv preprint arXiv:1706.05098*.

Ruff, Lukas et al. (2018). "Deep one-class classification". In: *International conference on machine learning*. PMLR, pp. 4393–4402.

Ruff, Lukas et al. (2021). "A unifying review of deep and shallow anomaly detection". In: *Proceedings of the IEEE*.

Sahu, Anit Kumar et al. (2018). "On the convergence of federated optimization in heterogeneous networks". In: *arXiv preprint arXiv:1812.06127*.

Samek, Wojciech, Thomas Wiegand, and Klaus-Robert Müller (2018). "Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models". In: *ITU Journal: ICT Discoveries - Special Issue 1 - The Impact of Artificial Intelligence (AI) on Communication Networks and Services* 1.1, pp. 39–48.

Samek, Wojciech et al. (2021). "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications". In: *Proceedings of the IEEE* 109.3, pp. 247–278. DOI: 10.1109/JPROC.2021.3060483.

Sandler, Mark et al. (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520.

Sanh, Victor et al. (2019). "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108*.

Sattler, Felix, Klaus-Robert Müller, and Wojciech Samek (2019). "Clustered Federated Learning". In: *Proceedings of the NeurIPS'19 Workshop on Federated Learning for Data Privacy and Confidentiality*, pp. 1–5.

Sattler, Felix, Klaus-Robert Müller, and Wojciech Samek (2020). "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13. DOI: 10.1109/TNNLS.2020.3015958. URL: https://doi.org/10.1109/TNNLS.2020.3015958.

Sattler, Felix, Thomas Wiegand, and Wojciech Samek (2020). "Trends and Advancements in Deep Neural Network Communication". In: *ITU Journal: ICT Discoveries* 3.1, pp. 53–63.

Sattler, Felix et al. (2019). "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication". In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852172. URL: http://dx.doi.org/10.1109/IJCNN.2019.8852172.

Sattler, Felix et al. (2020a). "On the Byzantine Robustness of Clustered Federated Learning". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8861–8865. DOI: 10.1109/ICASSP40776.2020.9054676. URL: http://dx.doi.org/10.1109/ICASSP40776.2020.9054676.

Sattler, Felix et al. (2020b). "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data". In: *IEEE Transactions on Neural Networks and Learning*

Systems 31.9, pp. 3400–3413. DOI: 10.1109/TNNLS.2019.2944481. URL: https://doi.org/10.1109/TNNLS.2019.2944481.

Sattler, Felix et al. (2021a). "CFD: Communication-Efficient Federated Distillation via Soft-Label Quantization and Delta Coding". In: *IEEE Transactions on Network Science and Engineering*, pp. 1–1. ISSN: 2327-4697. DOI: 10.1109/TNSE.2021.3081748. URL: https://dx.doi.org/10.1109/TNSE.2021.3081748.

Sattler, Felix et al. (2021b). "FedAUX: Leveraging Unlabeled Auxiliary Data in Federated Learning". In: *CoRR* abs/2102.02514. URL: https://arxiv.org/abs/2102.02514.

Sayood, Khalid (2017). *Introduction to data compression*. 5th. Morgan Kaufmann.

Schapire, Robert E. (1999). "A Brief Introduction to Boosting". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1401–1406.

Seide, Frank et al. (2014). "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns". In: *Fifteenth Annual Conference of the International Speech Communication Association*.

Seo, Hyowoon et al. (2020). "Federated Knowledge Distillation". In: *arXiv preprint arXiv:2011.02367*.

Sharkey, Amanda J. C. (1996). "On Combining Artificial Neural Nets". In: *Connect. Sci.* 8.3, pp. 299–314.

Shi, Shaohuai et al. (2019). "A distributed synchronous SGD algorithm with global Top-k sparsification for low bandwidth networks". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, pp. 2238–2247.

Shlezinger, Nir, Stefano Rini, and Yonina C Eldar (2020). "The Communication-Aware Clustered Federated Learning Problem". In: *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, pp. 2610–2615.

Shokri, Reza and Vitaly Shmatikov (2015). "Privacy-preserving deep learning". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, pp. 1310–1321.

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. URL: http://arxiv.org/abs/1409.1556.

Smith, Virginia et al. (2017a). "CoCoA: A General Framework for Communication-Efficient Distributed Optimization". In: *J. Mach. Learn. Res.* 18, 230:1–230:49. URL: http://jmlr.org/papers/v18/16-512.html.

Smith, Virginia et al. (2017b). "Federated multi-task learning". In: *Advances in Neural Information Processing Systems*, pp. 4424–4434.

Socher, Richard et al. (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1631–1642.

Sollich, Peter and Anders Krogh (1995). "Learning with ensembles: How overfitting can be useful". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 8, pp. 190–196.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Stich, Sebastian U, Jean-Baptiste Cordonnier, and Martin Jaggi (2018). "Sparsified SGD with memory". In: *Advances in Neural Information Processing Systems*, pp. 4447–4458.

Strom, Nikko (2015). "Scalable distributed DNN training using commodity GPU cloud computing". In: *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10,*

2015. ISCA, pp. 1488–1492. URL: http://www.isca-speech.org/archive/interspeech_2015/i15_1488.html.

Subramanyan, Pramod et al. (2017). "A formal foundation for secure remote execution of enclaves". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 2435–2450.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112. URL: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Taylor, Robin, David Baron, and Daniel Schmidt (2015). "The world in 2025-predictions for the next ten years". In: *10th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, pp. 192–195.

Tsuzuku, Yusuke, Hiroto Imachi, and Takuya Akiba (2018). "Variance-based Gradient Compression for Efficient Distributed Deep Learning". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sy6hd7kvM.

Vanhaesebrouck, Paul, Aurélien Bellet, and Marc Tommasi (2017). "Decentralized Collaborative Learning of Personalized Models over Networks". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. Vol. 54. Proceedings of Machine Learning Research. PMLR, pp. 509–517. URL: http://proceedings.mlr.press/v54/vanhaesebrouck17a.html.

Vapnik, Vladimir (2013). *The nature of statistical learning theory*. Springer science & business media.

Varma, Dandu Ravi (2012). "Managing DICOM images: Tips and tricks for the radiologist". In: *The Indian journal of radiology & imaging* 22.1, p. 4.

Voigt, Paul and Axel Von dem Bussche (2017). "The eu general data protection regulation (gdpr)". In: *A Practical Guide, 1st Ed.*

Wang, Hongyi et al. (2018). "ATOMO: Communication-efficient Learning via Atomic Sparsification". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 9872–9883. URL: https://proceedings.neurips.cc/paper/2018/hash/33b3214d792caf311e1f00fd22b392c5-Abstract.html.

Wang, Tongzhou and Phillip Isola (2020). "Understanding contrastive representation learning through alignment and uniformity on the hypersphere". In: *International Conference on Machine Learning*. PMLR, pp. 9929–9939.

Warden, Pete (2018). "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition". In: *arXiv preprint arXiv:1804.03209*.

Wen, Wei et al. (2017). "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1509–1519. URL: https://proceedings.neurips.cc/paper/2017/hash/89fcd07f20b6785b92134bd6c1d0fa42-Abstract.html.

Widrow, Bernard, Istvan Kollar, and Ming-Chang Liu (1996). "Statistical theory of quantization". In: *IEEE Transactions on instrumentation and measurement* 45.2, pp. 353–361.

Wiedemann, Simon, Klaus-Robert Müller, and Wojciech Samek (2020). "Compact and Computationally Efficient Representation of Deep Neural Networks". In:

IEEE Transactions on Neural Networks and Learning Systems 31.3, pp. 772–785. DOI: 10.1109/TNNLS.2019.2910073.

Wiedemann, Simon et al. (2020). "DeepCABAC: A Universal Compression Algorithm for Deep Neural Networks". In: *IEEE Journal of Selected Topics in Signal Processing* 14.3, pp. 1–15. DOI: 10.1109/JSTSP.2020.2969554. URL: http://dx.doi.org/10.1109/JSTSP.2020.2969554.

Wiegand, T. et al. (2003). "Overview of the H.264/AVC video coding standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7, pp. 560–576. DOI: 10.1109/TCSVT.2003.815165.

Wiegand, Thomas and Bernd Girod (2001). *Multi-frame motion-compensated prediction for video transmission*. Vol. 636. Springer Science & Business Media.

Wu, Bichen et al. (2019). "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747*.

Xing, Eric P et al. (2015). "Petuum: A new platform for distributed machine learning on big data". In: *IEEE Transactions on Big Data* 1.2, pp. 49–67.

Xu, Jie et al. (2021). "Federated learning for healthcare informatics". In: *Journal of Healthcare Informatics Research* 5.1, pp. 1–19.

Xu, Jinjin et al. (2020). "Ternary Compression for Communication-Efficient Federated Learning". In: *arXiv preprint arXiv:2003.03564*.

Xu, Kelvin et al. (2015). "Show, attend and tell: Neural image caption generation with visual attention". In: *International Conference on Machine Learning*, pp. 2048–2057.

Yang, Qiang et al. (2019). "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2, p. 12.

You, Xiaohu et al. (2021). "Towards 6G wireless communication networks: Vision, enabling technologies, and new paradigm shifts". In: *Science China Information Sciences* 64.1, pp. 1–74.

Yu, Hao, Sen Yang, and Shenghuo Zhu (2019). "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 5693–5700.

Yuksel, Seniha Esen, Joseph N. Wilson, and Paul D. Gader (2012). "Twenty Years of Mixture of Experts". In: *IEEE Trans. Neural Networks Learn. Syst.* 23.8, pp. 1177–1193.

Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals (2014). "Recurrent neural network regularization". In: *arXiv preprint arXiv:1409.2329*.

Zhang, Fengda et al. (2020a). "Federated Unsupervised Representation Learning". In: *arXiv preprint arXiv:2010.08982*.

Zhang, Wei et al. (2016). "Staleness-Aware Async-SGD for Distributed Deep Learning". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. IJCAI/AAAI Press, pp. 2350–2356. URL: http://www.ijcai.org/Abstract/16/335.

Zhang, Xiang, Junbo Jake Zhao, and Yann LeCun (2015a). "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28, pp. 649–657.

— (2015b). "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28, pp. 649–657.

Zhang, Xiangyu et al. (2018). "ShuffleNet: An extremely efficient convolutional neural network for mobile devices". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6848–6856.

Zhang, Zhengming et al. (2020b). "Benchmarking semi-supervised federated learning". In: *arXiv preprint arXiv:2008.11364*.

Zhao, Ying et al. (2019). "PDGAN: a novel poisoning defense method in federated learning using generative adversarial network". In: *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, pp. 595–609.

Zhao, Yue et al. (2018). "Federated learning with non-iid data". In: *arXiv preprint arXiv:1806.00582*.

Zhu, Yukun et al. (2015). "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27.

Zinkevich, Martin et al. (2010). "Parallelized stochastic gradient descent". In: *Advances in neural information processing systems*, pp. 2595–2603.