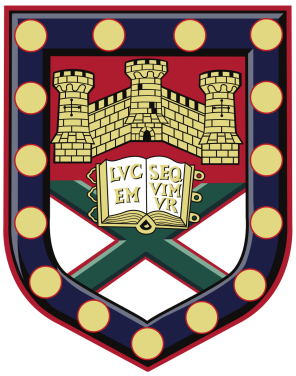
Federated Machine Learning in Edge Computing



Jed Mills

College of Engineering, Mathematics and Physical Sciences
University of Exeter

Submitted by Jed Mills to the University of Exeter as a thesis for the degree of *Doctor of Philosophy* in *Computer Science*.

I certify that all material in this thesis which is not my own work has been identified and that any material that has previously been submitted and approved for the award of a degree by this or any other University has been acknowledged.

Department of Computer Science August 2022

Acknowledgements

I'd like to thank everyone who supported and encouraged me over the years at university and throughout the PhD.

My supervisor and friend Jia for his reassuring guidance, helping me to learn the art and science of academic research and publishing.

The friends and collaborators from A1 for showing me the ropes and providing a relaxing working environment: Dongya, Haozhe, Jin, Rui, Siewei, Zhengxin.

All of the housemates who made living in Exeter a joy for the last seven years: Martha, James, Hannah, Inga, Jamie.

Josh, Rafe and Tom for being fantastic comrades with the ability to find the humour in any situation.

Finally, my love and gratitude to Emy, Dad and my wonderful Mum.

List of Publications

J. Mills, J. Hu, G. Min. "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT", *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986-5994, 2020.

J. Mills, J. Hu, G. Min. "Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 630-641, 2022.

J. Mills, J. Hu, G. Min. "Client-Side Optimisation Strategies for Communication-Efficient Federated Learning", *IEEE Communications Magazine*, vol. 60, no. 7, pp. 60-66, 2022.

J. Mills, J. Hu, G. Min, R. Jin, S. Zheng, J. Wang. "Accelerating Federated Learning with a Global Biased Optimiser", *IEEE Transactions on Computers* (accepted).

J. Mills, J. Hu, G. Min. "Faster Federated Learning with Decaying Number of Local SGD Steps", *IEEE Transactions on Parallel and Distributed Systems* (under 2$^{\text{nd}}$ round of review).

Z. Yu, J. Hu, G. Min, H. Xu, **J. Mills**. "Proactive Content Caching for Internet-of-Vehicles based on Peer-to-Peer Federated Learning", *IEEE International Conference on Parallel and Distributed Systems*, pp. 601-608, 2020 (invited paper).

R. Jin, J. Hu, G. Min, **J. Mills**. "Lightweight Blockchain-empowered Secure and Efficient Federated Learning at the Wireless Edge", *IEEE Transactions on Computers* (under review).

J. Wang, J. Hu, **J. Mills**, G. Min. "Federated Ensemble Model-based Reinforcement Learning", *IEEE Transactions on Parallel and Distributed Systems* (under review).

Abstract

Machine Learning (ML) is transforming the way that computers are used to solve problems in computer vision, natural language processing, scientific modelling, and much more. The rising number of devices connected to the Internet generate huge quantities of data that can be used for ML purposes.

Traditionally, organisations require user data to be uploaded to a single location (i.e., cloud datacentre) for centralised ML. However, public concerns regarding data-privacy are growing, and in some domains such as healthcare, there exist strict laws governing the access of data. The computational power and connectivity of devices at the network edge is also increasing: edge computing is a paradigm designed to move computation from the cloud to the edge to reduce latency and traffic.

Federated Learning (FL) is a new and swiftly-developing field that has huge potential for privacy-preserving ML. In FL, edge devices collaboratively train a model without users sharing their personal data with any other party. However, there exist multiple challenges for designing useful FL algorithms, including: the heterogeneity of data across participating clients; the low computing power, intermittent connectivity and unreliability of clients at the network edge compared to the datacentre; and the difficulty of limiting information leakage whilst still training high-performance models.

This thesis proposes new methods for improving the process of FL in edge computing and hence making it more practical for real-world deployments. First, a novel approach is designed that accelerates the convergence of the FL model through adaptive optimisation, reducing the time taken to train a model, whilst lowering the total quantity of information uploaded from edge clients to the coordinating server through two new compression strategies. Next, a Multi-Task FL framework is proposed that allows participating clients to train unique models that are tailored to their own heterogeneous datasets whilst still benefiting from FL, improving model convergence speed and generalisation performance across clients. Then, the principle of decreasing the total work that clients perform during the FL process is explored. A theoretical analysis (and subsequent experimental evaluation) suggests that this approach can reduce the time taken to reach a desired training error whilst lowering the total computational cost of FL and improving communication-efficiency. Lastly, an algorithm is designed that applies adaptive optimisation to FL in a novel way, through the use of a statistically-biased optimiser whose values are kept fixed on clients. This algorithm can leverage the convergence guarantees of centralised algorithms, with the addition of FL-related error-terms. Furthermore, it shows excellent performance on benchmark FL datasets whilst possessing lower computation and upload costs compared to competing adaptive-FL algorithms.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AI	Artificial Intelligence
BN	Batch Normalisation
CC	Cloud Computing
CE-FedAvg	Communication-Efficient Federated Averaging
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DNN	Deep Neural Network
dSGD	Distributed Stochastic Gradient Descent
EC	Edge Computing
ExpDQ	Exponential Dequantization
ExpQ	Exponential Quantization
FC	Fully-Connected
FedAvg	Federated Averaging
FedGBO	Federated Global Biased Optimiser
FL	Federated Learning
FLOPs	Floating Point Operations
GD	Gradient Descent
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IID	Independent and Identically Distributed
IoT	Internet of Things
MEC	Mobile Edge Computing
ML	Machine Learning
MTFL	Multi-Task Federated Learning
MTL	Multi-Task Learning
NC	Non-Converged
NLP	Natural-Language Processing
p2p	Peer-to-Peer
ReLU	Rectified Linear Unit
RPi	Raspberry Pi
SA	Sentiment Analysis
SGD	Stochastic Gradient Descent
UA	User Accuracy
UniDQ	Uniform Dequantization
UniQ	Uniform Quantization

Chapter 1

Introduction

Computers can be used to perform a huge range of complex and useful tasks. Programs can be written that allow people to connect over the Internet, manage huge quantities of information in databases, simulate and gain novel insights within science, engineering and medicine, control and record industrial machinery, and even beat expert human players in games such as chess. Nonetheless, the behaviour of these programs relies solely on the ingenuity of their designers.

Machine Learning (ML) is currently revolutionising many areas of science, engineering, business and society. It is concerned with building models that 'learn' from data to have good performance on an objective, rather than following pre-specified logic. ML systems require access to greater and greater quantities of data in order to improve their ability to generalise for previously-unseen inputs.

There are many sources of data with which to train ML models in the modern world. However increasing concern regarding the ownership and privacy of data motivates approaches for privacy-preserving ML. Alongside this, the power, connectivity and total number of devices on the Internet is increasing rapidly, providing opportunities to move expensive computation from the network core to the edge. The subject of this thesis is Federated Learning (FL): an exciting and recent paradigm for collaborative, distributed ML that takes place over the Internet in order to preserve the data privacy of users.

1.1 Machine Learning

Traditionally, computer programs take inputs and execute sets of precise formal instructions (written by programmers and engineers) to complete a predefined task, providing outputs for people and other machines. However for many interesting and useful tasks it is infeasible to define the logic to carry them out. Furthermore, scientists and engineers wish to gain novel insights into data so do not wish to explicitly specify program logic.

Machine Learning (ML) is a discipline within the broader field of Artificial Intelligence that builds models from data. These models 'learn' to improve their performance on an objective. This way, the precise relationship between program inputs and outputs do not need to be predefined, but rather emerge from the training dataset that is fed into the model. After training, the model is used for inference: processing new, unseen inputs to complete the desired task. The ability of the model to perform well on new examples is its *generalisation* ability. ML is an exciting and rapidly growing field with a huge range of applications, by no means limited to: reliably recognising elements within images and speech, making medical predictions with better efficacy than human experts, autonomously driving vehicles with greater

safety compared to human drivers, discovering new relationships within the social and natural sciences, and optimising resources within business and industry.

Due to state-of-the-art performance on a variety of tasks, a diverse range of prospective applications, and huge commercial potential, ML is experiencing intensive research efforts, with the total number of ML publications increasing by more than four times in the last 10 years [1]. ML is often categorised into the following broad sub-fields based on the kind of task to be solved. *Supervised Learning* predicts an output for a given input, requiring a set of existing input-output pairs, with examples include image classification and translation. *Unsupervised Learning* derives novel relationships within data and requires only inputs, with examples including data clustering and dimensionality reduction. *Reinforcement Learning* systems interact with a dynamic environment to maximise a reward, with examples including game-playing and robot control.

Alongside developments within ML domains and applications, there have been huge advances in the variety and complexity of the model used. *Nonparametric* models have a computational complexity that scales with the number of samples in the dataset, and include classic algorithms such as K-Nearest Neighbours and Gaussian Processes. *Parametric* models have a fixed number of parameters and computational complexity independent of dataset size, and include algorithms such as Logistic Regression and Linear Discriminant Analysis. Within the already rapidly-advancing field of ML, parametric Deep Neural Networks (DNNs) are receiving exceptionally intense research efforts. This is largely due to their ability to learn high-level features from raw datasets, reducing the need for feature engineering and leading to world-leading performance on a wide variety of tasks.

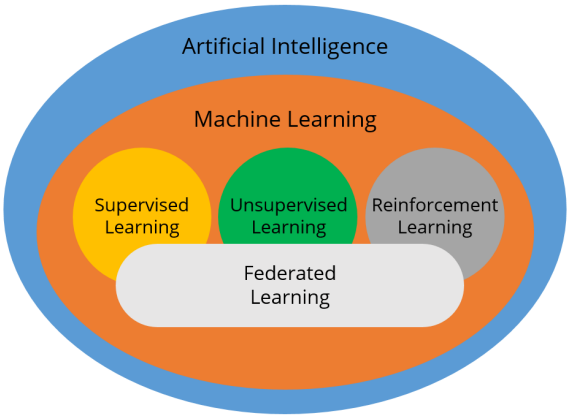


Figure 1.1: Conceptual relationship between Federated Learning and other fields within Artificial Intelligence.

Federated Learning (FL) is a recent development within ML for training models using datasets that are split across a large number of clients, without requiring those clients to share their data to any other party. It is a general framework that has been successfully applied to the different ML sub-fields, nonparametric and parametric models, with particular interest in supervised learning of large DNNs. Section 1.3 gives further details on FL, and Figure 1.1 conceptually places FL in relation to other disciplines within AI.

1.2 Edge Computing

Alongside developments in applications, computer hardware continues to see remarkable evolution. Due to miniaturisation, cost reduction, and improvements in speed, efficiency and connectivity, computers are becoming ever more ubiquitous and transforming every aspect of people's lives. One of the major uses of modern computers is to send and receive data over the Internet.

The number of devices connected to the Internet is therefore rapidly expanding, reaching over 12 billion as of 2022 [2]. User devices can range from increasingly powerful personal computers and smartphones to small, low-powered Internet of Things (IoT) devices and embedded controllers. The way that these devices are connected to the Internet also varies hugely: from super-fast fibre-optics and short-distance wireless protocols such as Wi-Fi to low-bandwidth long-distance networks like the Long-Term Evolution (LTE) standard. Subsequently, the computing and communications properties of user devices varies hugely.

Alongside the number of user devices, the quantity of data generated by them is also growing. By 2025, it is expected that the data stored by all Internet devices will reach 175 zettabytes [3]. The sources of this data are diverse, including but not limited to network traffic files, consumer media-content, social networking information, medical records, and sensor logs generated by the IoT.

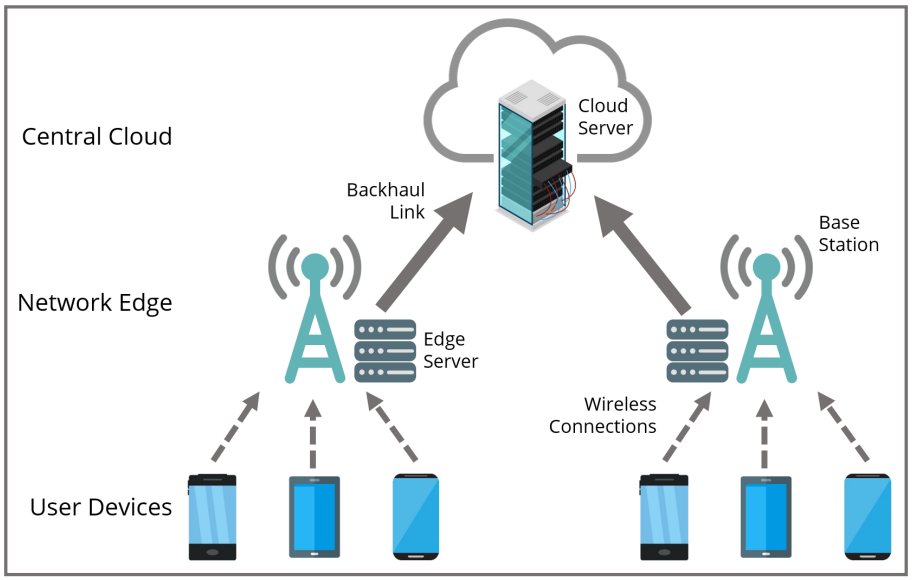


Figure 1.2: Edge Computing architecture.

User devices are connected to the Internet via the network edge, which includes wireless base stations and corporate network-entry points. The network edge in turn is connected to the Internet core, which interconnects large networks together, such as between countries. Figure 1.2 shows the basic architecture of the network edge. Cloud Computing (CC) is a paradigm for providing services such as compute and storage over the Internet. Powerful servers located near the network core carry out data storage, retrieval and expensive computation, and send the requested data to user devices. This means that even low-powered user devices with limited storage can operate as if they have access to almost unlimited storage and compute. Despite these benefits, CC has substantial drawbacks including high latency, poor performance when user bandwidth is low, high levels of network traffic as content is sent between from the cloud to the edge, and privacy concerns due to data storage

in remote locations.

Edge Computing (EC) aims to move this expensive computation and storage from the network core to the edge. These operations therefore take place close to the source of the data, alleviating the communication load within the Internet and improving latency whilst reducing the burden on the relatively small number of cloud servers [4]. Alongside the performance advantages of such a strategy, removing the need for data to be uploaded from the edge to the cloud provides enhanced privacy benefits to the data owners. Federated Learning is a recent distributed computing paradigm that moves Machine Learning from the cloud to the network edge and user devices.

1.3 Federated Learning

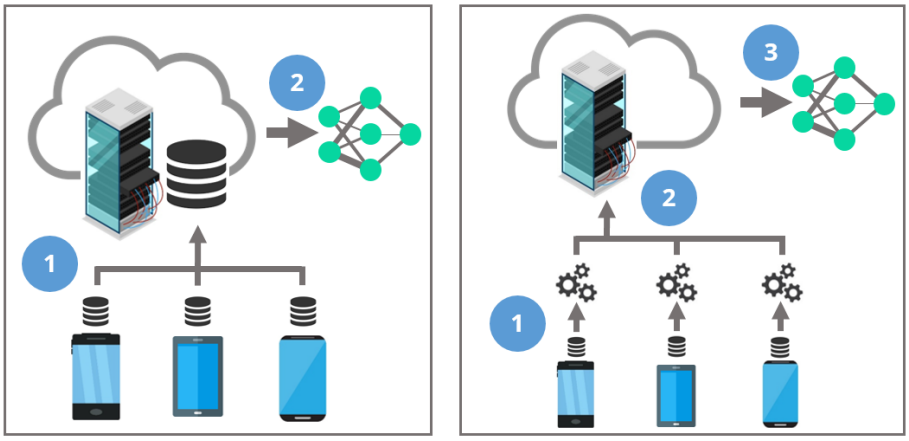
The generalisation ability of ML models benefits greatly from access to larger quantities of data. The huge amount of data generated by user devices therefore provides great opportunity for a range of ML applications. However, increasing public concern regarding data privacy and the establishment of landmark legislation such as the General Data Protection Regulation means that Internet users are increasingly unwilling or unable to share their data with parties that wish to train a model.

Traditionally, when organisations wish to train an ML model from user data, the data is uploaded by users and stored together in a datacentre. Training the model is then performed in the datacentre, using multiple high-powered compute nodes with high-bandwidth connections. The resulting model can then be used centrally, or shipped to edge servers or to user devices for inference. Large literatures exist both for datacentre-based training and for model inference at the edge [5], however Federated Learning (FL) is an emerging computing paradigm for distributed *training* of models at the network edge or user level whilst preserving data-privacy [6]. Each device that participates in FL (be it a wireless base-station equipped with compute and storage capabilities or a user's smartphone) is known as an FL *client*.

Despite the ever-growing quantity of data produced by users, the total data required to achieve good generalisation performance for state-of-the-art ML models (particularly DNNs) is typically much larger than that stored on any one client. Therefore the model produced by the federation of clients will in principle have superior generalisation performance than individual clients could achieve, as federated pool of data will be extremely large. FL performs additional processing on clients compared to simply uploading their data. The result of this processing is considered to be less privacy-sensitive than the underlying data, and further methods can be used to enhance and provide formal guarantees of level of privacy [7]. Figure 1.3 demonstrates the difference between the traditional cloud-based ML and FL approaches.

FL thus represents a marriage between ML and EC. It covers a range of scenarios characterised by the computing power, communications capabilities, total number, reliability, and amount of data owned by the participating clients. Broadly, in 'cross-device' scenarios a huge number of unreliable low-powered clients, each possessing a small quantity of data, collaborate to train the FL model. Alternatively, 'cross-silo' scenarios employ a smaller number of more reliable, more powerful clients each possessing larger quantities of data.

Real-world examples of cross-device deployments include millions of smartphones creating a next-work predictor [8], digital assistants jointly training a wake-word detector [9], and personal computers optimising Internet browser settings [10]. Exam-



(a) Centralised learning using edge data (b) Federated Learning using edge data

Figure 1.3: Cloud-based ML vs FL paradigms: (a) 1. clients upload their data to the cloud, 2. the cloud server performs ML centrally; (b) 1. clients perform local computation on their data, 2. clients upload the product of this data to the server, 3. the server aggregates the received information to produce an FL model.

ple of cross-silo deployments include hospitals jointly training a disease-prediction model [11], detection of money-laundering across financial institutions [12], and anomaly flagging in smart manufacturing plants [13]. A thorough overview of the FL scenario, its developments and its applications is given in [7]. Although FL can be deployed for a wide range of applications in many different environments, the following key features distinguish the FL scenario.

- *Distributed parallel computing:* there are a large number of clients collaboratively training a model, with high-latency and low-bandwidth connections compared to training in a datacentre, leading to long training times. FL clients are coordinated by a central server. The connections to the server are unreliable due to user behaviour and connectivity, and the compute and communication abilities of clients can be very non-uniform. These characteristics present significant systems and protocol challenges for reliable FL algorithms.
- *Strict data privacy:* the data stored on each client cannot be sent to any other client or the coordinating server. Furthermore, minimal information about the data distribution on each client should be shared during the FL process in order to maximise data privacy. Ideally, the information sent to the server by any client should not uniquely identify it. Reducing information and privacy leakage is a significant challenge whilst still training an FL model with good performance.
- *Heterogeneous client data:* as the client devices and their owners have different behaviour and environments, the data generated by clients is highly heterogeneous and non-Independent and Identically Distributed (non-IID). Furthermore, the distribution of data on clients can change over time as user behaviour changes. Non-IID client data significantly increases the difficultly with which a high-quality FL model can be trained.

1.4 Research Challenges & Objectives

Despite similarities to distributed parallel computing in the datacentre, the unique privacy-related and EC features of FL present a novel and challenging field for

researchers. New algorithms and systems must be designed that: exhibit good generalisation performance across clients with heterogeneous data, display fast convergence due to the slow communication and computation properties of the network edge compared to the datacentre, have low compute and communication overheads due to client device limitations, and are resilient to unpredictable or even malicious clients.

All of these goals must be achieved whilst adhering to FL's strict data-privacy requirements. FL has the potential to drastically change how organisations train models from user data, opening new directions for innovative privacy-focussed and ethical applications which improve users' trust in the systems that utilise their data.

The aim of this thesis is to develop novel techniques and algorithms for improving the FL process and making it more practicable for real-world deployments. Namely, these contributions are designed to achieve the following broad objectives:

1. To reduce FL's communication and computation overheads in order to meet the unique hardware constraints of the EC environment.
2. To improve the generalisation performance of FL models given across heterogeneous client datasets and hence make participation more attractive to users.
3. To accelerate the convergence rate of the FL model and hence reduce the time taken to complete an FL deployment, making it more practical for real-world use.

1.5 Contributions & Organisation of Thesis

To achieve the research objectives set out in Section 1.4, the remaining chapters of this thesis make the following original contributions.

Chapter 2 provides a detailed and formal description of the FL process, and surveys important developments within FL that are built upon in the other chapters. It also overviews works in the topics of EC and datacentre-based training that are relevant to this thesis. Current challenges to the state-of-the art are also presented.

Chapter 3 develops methods for improving FL's communication-efficiency. This is achieved through the novel application of adaptive optimisation in combination with two new techniques for compressing the information uploaded by FL clients to the coordinating server. These approaches significantly reduce the total quantity of data communicated and total time taken for an FL model to reach a target accuracy.

Chapter 4 proposes a new personalisation strategy to boost model performance on heterogeneous client data. Personalised models that are tailored to each non-IID client dataset are trained whilst still benefiting from FL through the use of a mixture of private and federated DNN layers. Different combinations of private and federated elements are thoroughly investigated and show excellent improvements in convergence rate and generalisation.

Chapter 5 explores the principle of decreasing the amount of local work that clients perform during the training process, which can heavily improve FL's computational efficiency. A theoretical analysis suggests that this approach can drastically reduce the amount of time taken for the model to reach a given error, and motivates three ways in which the amount of local computation can be decayed during training.

Chapter 6 presents a novel algorithm for applying adaptive optimisation to FL. By utilising a statistically-biased adaptive optimiser that is applied globally on all clients, the convergence rate, generalisation performance, computational cost and communication overheads of FL are substantially reduced when compared to similar algorithms.

Chapter 7 concludes the thesis. After reviewing the technical contributions made in the previous chapters, current challenges to the state-of-the-art and future directions within the field of FL are discussed.

Chapter 2

Background & Related Work

This chapter provides a thorough background on FL as an ML optimisation problem and describes the working environment from a computer-systems perspective. It then presents the important Federated Averaging algorithm and explores the learning challenges related to it. Works from the FL and distributed learning literature related to the subsequent chapters are then surveyed.

2.1 The Federated Learning Objective

Organisations use FL to create a model from client data without accessing the training data, in order to preserve user privacy. After it is trained, the FL model may be used for inference in the cloud or shipped to devices at the edge [7].

In parametric ML, the objective is to minimise the loss of a model $\boldsymbol{x} \in \mathbb{R}^d$ over the whole distribution of data that it will be applied to (the *true risk*). However, ML models only have access to a limited number of examples from the true distribution (the training samples). Therefore only the *empirical risk* (error on training set) can be minimised, and it is assumed that the empirical risk corresponds to the true risk. The performance of a model on unseen examples is its *generalisation performance*, and the difference between the training and generalisation performance is the *generalisation gap*. The generalisation gap can usually be reduced by using a larger training dataset.

There are a large number of clients connected to the Internet, each possessing a small amount of training data. It is this data that cannot be accessed by the FL organiser, in order to preserve data privacy. However, *some* kind of information must be uploaded by clients to collectively train a model. This leads to a trade-off between utility and privacy. If all clients were to upload their data to a server for centralised training, the generalisation gap of the resulting model would likely be very small due to a huge data pool, but there would be little privacy guarantee. Alternatively, if all clients trained ML models independently (which would guarantee data-privacy), the generalisation performance of the individual models would be poor. FL therefore represents a 'good solution' in this privacy-utility tradeoff.

The FL objective is to train a *global model*. This model should minimise the *global loss function* F, which is the expected loss over client objectives, namely:

$$F(\boldsymbol{x}) = \mathop{\mathbb{E}}_{i} \left[f_i(\boldsymbol{x}) = \frac{1}{n_i} \sum_{j=1}^{n_i} f(\boldsymbol{x}; \xi_{i,j}) \right], \tag{2.1}$$

where f_i, n_i, and $\{\xi_{i,j}, \cdots, \xi_{i,n_i}\}$ denote the average loss, total number of samples, and individual training samples on client i, respectively. f is the loss function that

clients use to train $\boldsymbol{x}$. Intuitively F is the expected loss over all samples and all clients. The individual losses f_i are presented as a sum and the global loss F as an expectation to emphasise that in FL there are a large number of clients, each possessing a small number of local samples. This objective is analogous to that used when training a centralised model on pooled data (as in the datacentre). As such, the distributed ML literature has some overlap with the FL literature and is surveyed below.

If the client loss functions are convex, the *local minimiser* f_i^* of each client is defined as the point on the objective landscape such that:

$$f_i^* \leq f_i(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathbb{R}^d. \tag{2.2}$$

Due to different user behaviour and environments, client devices are assumed to have heterogeneous data distributions. The samples ξ_i on to each client also cannot be shared with any other client in order to preserve data-privacy. Therefore, the local minimiser for any two clients i and j are not necessarily the same: $f_i^* \neq f_j^*$.

If all the data across all clients is considered as the total federated dataset, then it is non-Independently and Identically Distributed (non-IID) across the clients. Non-IID data is one of the major obstacles for FL because it usually harms the final performance of the trained model, but is one of the distinguishing features of FL. For supervised learning problems (the focus of many FL works), there are many ways in which client data can be heterogeneous: non-IID features or labels, different underlying mapping between features and labels, non-uniform numbers of samples, and more [14]. For highly nonconvex loss surfaces such as those of DNNs (likely the most popular type of model used in FL), the landscapes and minimum points are also non-identical due to heterogeneous client data and the definition of the local loss (2.1).

Parametric models are typically trained using variants of Gradient Descent (GD). GD requires a loss function that is differentiable with respect to the parameters of the model ($\boldsymbol{x}$). With a loss defined over training samples as in (2.1), the gradient of the loss can be evaluated iteratively and the parameters of the model updated by a small value. Thus, the trained model descends to a low point on the *loss surface*. Traditional parametric models such as Linear Regression typically have convex loss surfaces, whereas modern DNNs have highly nonconvex surfaces [15], making the optimisation problem more difficult.

For a given starting model $\boldsymbol{x}_0 \in \mathbb{R}^d$ and loss function f, the recurrence relation for GD is given by:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t), \tag{2.3}$$

where η is the *learning rate*. GD is performed in steps until the model error $f(\boldsymbol{x}_t)$ reaches a desired value.

As the loss is defined over the whole training dataset (2.1), computing $\nabla f(\boldsymbol{x})$ at every iteration is prohibitively costly. A stochastic estimate of the gradient at $\boldsymbol{x}_t$ can instead be computing using only a single sample:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t; \xi_t), \tag{2.4}$$

The stochastic gradient in (2.4) is equal to true gradient in expectation: $\mathop{\mathbb{E}}_{i}[\nabla f(\boldsymbol{x}; \xi_i)] = \nabla f(\boldsymbol{x})$, but the estimate usually has high variance (increasing with the size of the dataset). This variance can slow convergence substantially. For smooth convex objectives and appropriately η, the error of GD after t iterations is bounded by $\mathcal{O}(1/t)$, compared to $\mathcal{O}(1/\sqrt{t})$ when Stochastic Gradient Descent (SGD) [16].

A compromise between variance and per-iteration cost can be found by computing the stochastic gradient over a *minibatch* of samples (typically tens to hundreds):

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{1}{B} \sum_{b=1}^{B} \nabla f(\boldsymbol{x}_t; \xi_{t,b}), \tag{2.5}$$

where B is size of the minibatch. In practice, variants of minibatch SGD are usually used for training state-of-the-art parametric models like DNNs [17, Chapter 8].

In the datacentre, Distributed-SGD (dSGD) is one of the most popular algorithms for training large models in a distributed fashion. At each step of dSGD, compute-nodes calculate a small minibatch gradient independently and sends these gradients to a parameter-server. The parameter-server averages these to produce a larger-batch gradient and applies it to the model. This new model is sent to the compute-nodes for the next iteration.

Despite the similarities of FL to distributed datacentre training (in that a number of nodes/clients train a model together), the design of dSGD makes it unsuitable for the FL scenario. Given a total number of model parameters d and clients c, the data communicated by dSGD over t iterations is $\mathcal{O}(2cdt)$. Contemporary models typically require millions of dSGD iterations to converge to a desired error. Communicating this much data over the Internet would be a large burden. Furthermore, FL clients are assumed to have low-bandwidth connections to the coordinating server (particularly for upload, given the architecture of the network edge) [18]. Thus the time taken for each iteration of dSGD in FL would be large and the total runtime inordinately long. It is also infeasible to require all FL clients to participate at each step: for real-world cross-device deployments, the fraction of clients available for training at any point in time can be as low as 0.2% [8]. These factors motivate algorithms that can converge in fewer rounds of communication and are robust to low client participation rate, as discussed below.

2.2 Constraints of the Network Edge

In the datacentre, compute-nodes have high-bandwidth interconnects are used to train large-scale models, particularly DNNs. To further accelerate training, nodes are usually equipped with powerful Graphics Processing Units (GPUs). GPUs have a highly-parallel architecture that allows them to perform the same operation on large blocks on memory concurrently. Large matrix multiplications and other parallel operations are required to propagate values through DNNs, so GPUs show significant speedup compared to traditional Central Processing Units (CPUs) for this task. Furthermore, the next generation of datacentre-based ML is likely to include Tensor Processing Units (TPUs), which are accelerators even further specialised for DNN training [19].

However the environment of the network edge is far more constrained. User devices are connected to the edge using a variety of connection types, ranging from coaxial cable and fibre-optics, to Wireless Local Area Networks (WLAN) and cellular networks. Despite improvements in wireless cellular networks (from 1G to 5G) for greater and greater bandwidth in order to support multimedia-content delivery, the bandwidth of even the latest 5G networks is a small fraction of the bandwidth that compute-nodes have between each other in the datacentre [18]. Due to the huge number of users connected to the Internet, it is impossible to enforce a single communications technology amongst all devices, and it takes decades to shift the

predominant technology in use [20]. Furthermore, due to the architecture of the network edge the upload bandwidth of user and edge devices is typically much lower than the download bandwidth. This asymmetry poses a challenge for many FL algorithms, which typically require uploading and downloading large quantities of data between the clients and coordinating server many times.

Alongside the constraints on communications, client hardware is very diverse but generally of much lower power compared to the datacentre. In some FL deployments, IoT devices connect to and store their data on local edge servers. These EC servers can have a range of computing and storage hardware, with some modern servers equipped with GPUs [21]. In others, the client-base includes EC servers and individual devices at the user devices (see Figure 1.3). Some FL researchers have even proposed performing local computation on embedded IoT devices themselves [22]. Clients are also unreliable and can join and leave the federation arbitrarily. For example, in previous deployments smartphone clients could only participate in the training when charging and connected to an unmetered wireless network [8].

These factors generally mean that FL algorithms should attempt to minimise the computational and communication costs of training. It is usually assumed in FL that the time spent waiting for straggling (slow) clients and the time spent uploading the FL data dominate the runtime. Compression and other communication-reduction strategies are an active research topic within FL [7], and strategies for lowering the local computation cost are also emerging.

2.3 Federated Averaging

To better utilise resources and decrease the runtime of dSGD in the datacentre, multiple steps of SGD can be performed on compute-nodes between communications. Instead of sending model gradients to the parameter server, nodes perform several steps of SGD and send the resultant models to the parameter server. The server averages the nodes' models to produce the next iteration's model. This approach is usually termed local-SGD [23]. Despite reducing the communication overheads of distributed training, local-SGD is still unsuitable for the FL scenario as it requires all nodes to participate in every round of communication. Theoretical analyses of local-SGD also typically assume that the number of samples per node is much larger than the number of nodes, which is not the case in FL (2.1).

Algorithm 1: Federated Averaging (FedAvg) [24]

input: initial global model $\boldsymbol{x}_0$
for round $r = 1$ **to** R **do**
  select round clients $\mathcal{C}_r$
  for client $c \in \mathcal{C}_r$ in parallel **do**
    download global model $\boldsymbol{x}_r$
    for local SGD step $k = 1$ **to** K **do**
      $\boldsymbol{x}^c_{r,k} \leftarrow \boldsymbol{x}_r - \eta \frac{1}{B} \sum_{b=1}^{B} \nabla f(\boldsymbol{x}^c_k, \xi^c_{k,b})$
    end
    upload local model $\boldsymbol{x}^c_{r,K_r}$ to server
  end
  update global model $\boldsymbol{x}_{r+1} \leftarrow \frac{1}{|C_r|} \sum_{c \in C_r} \boldsymbol{x}^c_{r,K_r}$
end

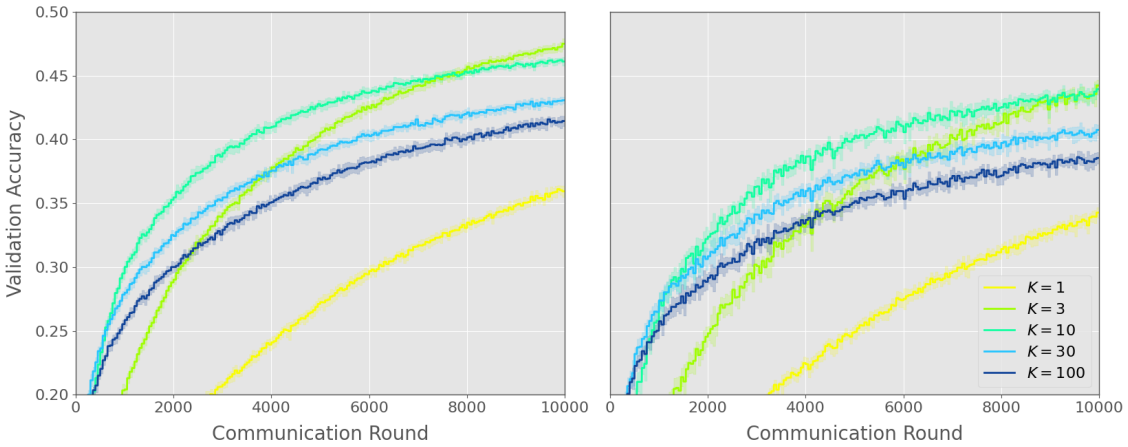


Figure 2.1: The effect of increasing the number of local SGD steps (K) performed during FedAvg when training a Convolutional Neural Network on the CIFAR100 dataset. Data is either IID (left) or non-IID (right) across 500 clients, with 5 clients participating per round. Curves and shaded regions represent mean and 95% confidence intervals for the validation accuracy over 10 random trials.

The Federated Averaging (FedAvg) algorithm was designed to alleviate the need for all clients to participate in every round of communication [24]. As the seminal FL algorithm, FedAvg is highly influential (being the basis of myriad extensions and improvements), and is likely the algorithm that is most used in current real-world deployments.

As such, FedAvg is presented in Algorithm 1. In each communication round, a subset of all clients at the network edge download the global model from the cloud-based coordinating server (line 5), perform K steps of minibatch SGD on their local datasets (lines 6-8), and upload their models to the server (line 9). The server then averages the received models to create the next round's global model (line 11). Typically, FedAvg is run for a given number of communication rounds or a pre-specified time budget, which can be up to weeks in reality [8]. When algorithmically described and theoretically analysed, in each round of FedAvg a subset of all clients are selected uniformly at random to participate (line 3). However in reality, in each round the server waits until a given number of clients upload their models before starting a new round. Also, the amount of work each client performs is sometimes allowed to vary depending on the size of their local dataset [24].

Whilst not providing any formal guarantee of the level of privacy provided, FedAvg does not require clients to share their training data with any other party. Further techniques such as differential privacy and homomorphic encryption can be used to enhance the level of privacy of FedAvg [7].

FedAvg's *communication-efficiency* is the number of communication rounds required to reach a target model error (or generalisation performance). This is different to its *iteration complexity*: the total number of SGD iterations. For a fixed number of local steps K and communication rounds R, the total number of iterations (disregarding the parallel work performed by clients) is given by $T = KR$. FedAvg's communication-efficiency can be improved by increasing the amount of local work performed by clients.

Figure 2.1 shows the validation performance of a Convolutional Neural Network (CNN) trained using FedAvg on the CIFAR100 dataset. In Figure 2.1 (a), the data is IID amongst 500 total clients. In Figure 2.1 (b), the data is partitioned in a

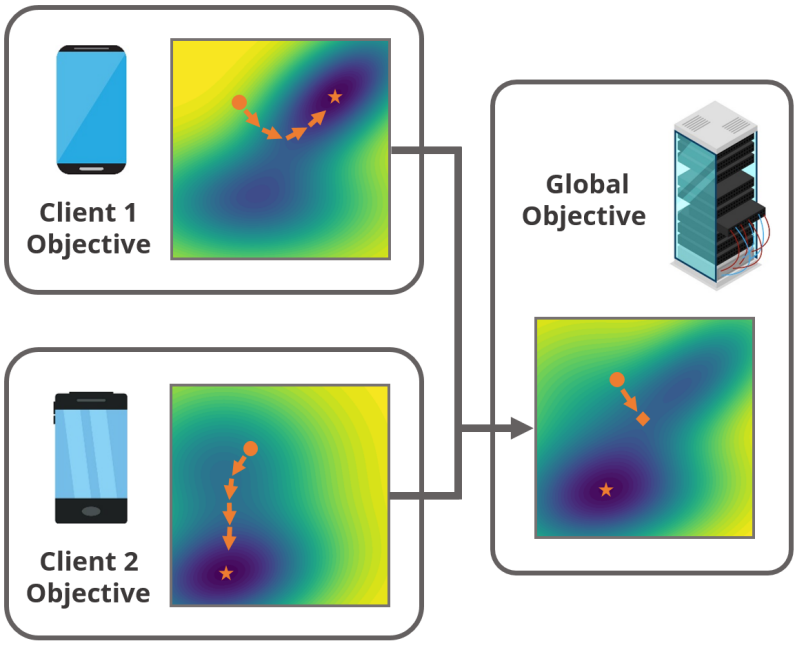


Figure 2.2: Demonstration of client-drift with two non-IID clients. Each client has a local loss function (contour plots), with the global loss being the average of client losses. The global model at the start of the round is represented by the circle. During local training, the clients' models move towards their minima (stars). The next round's global model (diamond) is the average of client models. Note that the average of client minimisers is not the same as the minimiser of average client losses (star on global loss surface).

non-IID fashion using the class labels, mimicking heterogeneous real-world clients [25]. In each communication round, 5 of the clients are selected to participate uniformly at random. For both data partitions, increasing the number of local SGD steps K increases the initial rate of convergence. However for large values of K, the final performance of the model is harmed. Furthermore, large values of K give diminishing returns even for improving the initial convergence rate.

The convergence of the global model is slower and reaches a lower final validation accuracy in Figure 2.1 (b). Inspecting the training curves, the variance in validation accuracy between rounds is higher for non-IID clients (resulting in noisier curves). The reason behind this is the heterogeneous data across clients leading to *client-drift* [26]. When data is non-IID, the surfaces of the client losses (f_i) have different shapes and minimum points. Figure 2.2 shows an example 2-dimensional nonconvex loss surface for two FL clients (corresponding to a model with 2 parameters). Lower loss is shown as dark blue, with orange stars representing the minimum points. The global loss surface is defined as the expectation over client losses as per (2.1).

The orange circles represent the starting point of a round of FedAvg. This point is far from each client's local minimum and from the global minimum. During $K = 4$ steps of local gradient descent the client models move (*drift*) towards their minimisers (orange arrows on client surfaces), which are not identical due to non-IID data. The next round's global model is the average of the two client models (orange diamond); the average of the client minimisers is not the same as the minimiser of the expected loss.

This simple 2-dimensional example serves to illustrate the challenge posed by client-drift. For extremely high-dimensional models such as DNNs, where clients perform a few steps of SGD (but not enough to converge to a local minima within

a single local update), client-drift harms the global convergence rate and best performance that the model can reach, as demonstrated by Figure 2.1. Previous works have derived bounds on the extent of client-drift, showing that it is a function of client data heterogeneity, the learning rate η and number of local steps K [25, 26, 27].

Client-drift is an important challenge to address within FL to allow algorithms performing multiple local steps to become more communication-efficient. Many of the related works surveyed below make strides towards counteracting client-drift.

2.4 Related Work

Due to FL's huge potential for privacy-preserving ML, there has been an intense research effort to solve many related problems since its inception in 2016. In this section, many of the major works relevant to the subsequent chapters are surveyed.

2.4.1 Theoretical Analysis of FedAvg

There has been significant work towards improving the theoretical understanding of FedAvg and related algorithms. Proving the theoretical properties of FedAvg is an important steps towards understanding its true behaviour and motivating new algorithms that leverage the theoretical properties.

In the first analysis of FedAvg for non-IID clients with partial participation, Li *et al.* [28] proved an iteration complexity of $\mathcal{O}(1/t)$ for smooth and strongly-convex client objective functions. Their analysis suggested that an optimal number of local steps K exists to maximise communication-efficiency, and the need to decrease the learning rate η during training to reach arbitrarily low error. Dominant convergence of FedAvg (in terms of iteration complexity) compared to dSGD is an open research problem for all but quadratic objectives. Karimireddy *et al.* [26] added a server learning rate to FedAvg to prove its convergence on nonconvex objectives, which has been later shown to improve FedAvg's iteration complexity [29]. Analysing FedAvg using quadratic objective functions has been beneficial for gaining gain insights into the trade-off between convergence rate and final model error [30], and for determining the influence of client gradient-dissimilarity on convergence [31].

Malinovsky *et al.* [32] generalised local-SGD methods to generic fixed-point functions (i.e., any process that converges to a stationary point, including SGD) to analyse the effect of K on the ϵ-accuracy. As with FedAvg, using larger K means that local-SGD cannot reach as cannot descend as close to the global minima. Linear speedup (in terms of the number of participating clients) of the iteration complexity for convex and nonconvex objectives have also been achieved [33, 34].

Chapter 5 of this thesis extends previous analyses to theoretically study the convergence of FedAvg when using a decaying value of K. This approach can reduce the total amount of real-time taken to reach a given model error and reduces the total computational cost of FedAvg, which beneficial for energy-efficiency [35].

2.4.2 Novel Optimisation Strategies

Several approaches have been proposed to accelerate the convergence rate of FedAvg and reduce client-drift. This is useful both for reducing FedAvg's runtime and improving the final performance of the FL model.

Proximal-based methods reduce client-drift by adding a term to the client loss functions discouraging client models from moving too far from the current global

model. The authors of FedProx [36] proved the intuition that the proximal coefficient should be increased as client data becomes more non-IID. FedSplit [37] extended FedProx by splitting the local proximal update into a two-step procedure, proving that the minimum points of the client objectives are the same as the global minimum point for convex objectives.

The global model updates during FedAvg can have high variance due to the minibatch-SGD used on clients and from sampling only a very small fraction of the non-IID clients per round. SCAFFOLD [26] uses *control variates* to add Stochastic Variance-Reduced Gradients (SVRG) at both the global and client level to mitigate the impact of client-drift and reduce variance. Control variates have been subsequently applied in multiple other algorithms [27, 38].

Adaptive optimisation strategies have long been used to accelerate training in convex and nonconvex optimisation, especially for large DNN models where computing higher-order statistics of the gradient is prohibitively costly [17, Chapter 8]. Previous authors have applied adaptive optimisation during the server step of FedAvg (Algorithm 1, line 11) [9, 25], which can accelerate FedAvg without increasing the per-round communication or computation cost for clients. Alternatively, adaptive optimisation can be applied during the client update, with the adaptive optimisers averaged alongside the client models at the each of each round [39, 40], or reset at the beginning of each round [41], or incorporated with adaptive server optimisation [42]. Another approach is to keep a set of global optimiser statistics that are kept fixed during the client loop and updated at the end of each round [27].

In this thesis, Chapters 3 and 5 both propose novel optimisation methods for FL. These methods accelerate the convergence rate of FedAvg, thus improving its communication-efficiency and reducing its computational cost and runtime.

2.4.3 Communication-Reduction in FL

As the FL process can occur over slow wireless connections and place a large burden on edge and backhaul links due to a huge number of clients, many works have also proposed reducing the per-round communication cost of FL. Communication-reduction is particularly important when large DNNs (with up to billions of parameters for state-of-the-art models) are trained.

There is a significant literature for compressing the models uploaded by clients during FedAvg, which is particularly important due to the asymmetric bandwidths of the network edge. An early work [43] empirically studied different approaches for dropping predetermined values from the uploaded models. There are also multiple works using *sparsification* (dropping all weights updates with small magnitudes) and *quantisation* (representing weight updates with fewer bits than standard 32-bit floating-point) [44, 45, 46, 47]. These compression strategies can achieve huge compression ratios without breaking the training procedure.

Models can also be uploaded with novel communication schemes. One approach is aggregating some layers of a federated DNN less frequently based on their position within the model [48]. Another is *hierarchical FL*, where clients aggregate with edge servers with a higher frequency than the cloud server [49, 50, 51].

Some works consider communication reduction for the physical layer. Over-the-Air computation allocates several clients to the same wireless channel to allow their models to additively interfere. This reduces the number of wireless channels needed to support a large number of clients, and is an emerging sub-field of FL [52, 53, 54].

As part of an algorithm that improves the communication-efficiency of FedAvg

through adaptive optimisation, Chapter 3 of this thesis proposes two novel strategies for compressing the model and optimiser values uploaded to the cloud server.

2.4.4 Personalised FL

FedAvg trains a single global model by performing rounds of SGD on clients followed by model averaging. However a core challenge within FL are clients with heterogeneous data. Therefore if the global model is distributed to clients after the training is complete, it will have non-uniform performance across them. Personalised FL methods are designed to improve the performance of the model(s) trained across non-IID clients [55].

A simple approach for personalisation is *fine-tuning:* clients download the final global model trained by FedAvg and perform a few steps of SGD on their data before local inference. Fine-tuning has been shown to improve the average generalisation performance across clients [56, 57]. Some works have extended this idea by applying Model-Agnostic Meta Learning (MAML) to FL, which alters the client objective functions to make the global model more easily adapted post-FedAvg [58, 59, 60]. However, there is debate regarding the true benefit of these more complex approaches compared to simple fine-tuning [61].

Another strategy is to train a mixture of global and local models during FL. This can be achieved by expressly designing the mixture in the optimisation problem [62], by reformulating FedAvg using Moreau envelopes [63], or by regularising between an explicit global and local objective [64].

FL can also be re-framed as a Multi-Task Learning (MTL) problem, where the heterogeneous clients represent a similar learning task to be jointly optimised. This has allowed authors to propose a primal-dual framework to optimise convex objective functions [65], formulate clients' non-IID data as mixtures of underlying global distributions [66], or by clustering users with similar (inferred) datasets [67].

Chapter 4 of this thesis proposes a new algorithm for training DNNs in federated MTL through the use of a mixture of private and global layers. The superior generalisation performance for non-IID clients is demonstrated against several other state-of-the-art algorithms [60, 63].

2.4.5 FL in Dynamic Edge Environments

Many works consider varying the amount of computation performed by clients depending on their available resources such as wireless bandwidth, battery power, CPU speed and current utilisation. These approaches variously improve FL's communication-efficiency, runtime, total energy or total data cost.

Scheduling of wireless clients is of strong interest in FL. In [38] the authors modelled the convergence of their algorithm in a time-sharing wireless-edge environment, incorporating factors such as client bandwidth, transmission power and CPU frequency to jointly minimise energy consumption and training time. [68] analysed scheduling policies factoring in the Signal-to-Noise ratio of wireless channels. Several works also schedule the amount of work clients perform based on their wireless bandwidth [69, 70, 71, 72], and it is common practice when using smartphone clients to limit participation to those that are connected to an unmetered network, charging and idle [8].

Asynchronous algorithms can be used to improve resource utilisation in the federation. These algorithms allows clients to download and upload their models at arbitrary times, removing the constraint of waiting for a sufficient number of clients

to complete each round. While alleviating the effect of stragglers, the performance of the global models produced by asynchronous FL algorithms is usually lower than synchronous ones (such as FedAvg). Asynchronous algorithms are also have worse theoretical guarantees [73, 74, 75].

The algorithms and techniques proposed in this thesis target a variety of domains and client hardware. In Chapters 3 and 4, experiments using wireless FL testbeds evaluate the performance of the proposed algorithms under realistic EC conditions. Furthermore, the asymmetric communications properties of the wireless edge are incorporated into the algorithm design in Chapters 3 and 6.

2.4.6 Datacentre-Based Developments

Distributed training in the datacentre shares similarities with the FL scenario in that there are multiple clients (nodes) collaboratively training a model. Therefore, some recent developments within this field are relevant to the chapters of this thesis. dSGD and local-SGD methods are commonly used in the datacentre due to the reliability and power of the compute nodes.

Woodworth *et al.* [76] proved for quadratic objectives that the iteration complexity of local-SGD dominates that of dSGD for quadratic objectives. However, even for more general convex problems local-SGD does not dominate. Similarly, Wang *et al.* [77] unified the analysis of various algorithms related to local-SGD with different communication topologies and achieved state-of-the-art results for some settings. Empirical studies have shown that generalisation performance can be improved by switching from dSGD to local-SGD in the later stages of training [78] or through the use of *extra-gradient* methods that compute gradients after a step of SGD on nodes [79]. Topics such as Transfer Learning [80] and MTL [81] typically studied in the datacentre setting have also been applied to FL [82, 65].

Furthermore, there have been communication-reduction strategies tailored to the datacentre that can be applied to FL. Frameworks that accumulate gradients dropped using gradient sparsification [44] have been extended to the FL scenario [45]. Similarly, model pruning [83] is a popular method developed for model sparsification in the datacentre that has inspired works for communication-reduction in FL [84, 85, 86].

Developments in the distributed-training literature can therefore be relevant for and influence progress within FL. The methods developed in Chapter 5 and Chapter 6 of this thesis could in turn be used in datacentre settings due to their broad applicability.

2.5 Chapter Summary

This chapter provided a primer on the FL optimisation problem and the physical characteristics of the FL environment. The primary algorithm for training parametric models in FL was described alongside its behaviour when deployed on heterogeneous clients. To place the contributions of this thesis within the wider FL literature, a broad survey of related sub-topics was then presented. The subsequent chapters make technical contributions that build and improve upon this existing body of work.

Chapter 3

Communication-Efficient Federated Learning for Wireless Edge Computing in IoT

This chapter proposes a new algorithm, Communication-Efficient Federated Averaging (CE-FedAvg), designed for the EC scenario where edge servers store data from nearby IoT devices. The huge number of IoT devices worldwide provide much opportunity for FL using their data. Although FedAvg [24] is commonly used for FL, it can suffer from a large number of rounds to convergence when client datasets are non-IID and high communication costs per round stemming from the need to transmit entire models between clients and server (see Section 2.3). FL is typically used to train large DNN models which can have up to billions of parameters, meaning that communication can present a significant bottleneck for FedAvg. This is especially true for model upload, considering the asymmetric upload and download bandwidths of the network edge. CE-FedAvg uses distributed Adam optimisation [87] to greatly reduce the number of rounds required for the global model to converge, along with two novel compression techniques utilising sparsification (dropping low-magnitude weight deltas), quantization (representing weight deltas as 8-bit integers rather than 32-bit floats), and Golomb encoding [88] (to represent weight delta indexes with fewer bits), that are designed to reduce the per-round upload cost. Extensive experiments are performed with the MNIST and CIFAR-10 datasets, IID and non-IID client data, varying numbers of clients and client participation, and compression rates. These show that CE-FedAvg can converge to a target accuracy in up to 6× fewer rounds compared to similarly-compressed FedAvg while uploading up to 3× less data, and is more robust to aggressive compression. Experiments on an edge-computing testbed using Raspberry Pi clients also show CE-FedAvg is able to reach a target accuracy in up to 1.7× less real time than FedAvg.

3.1 CE-FedAvg

The FedAvg algorithm has a single master model that is an aggregate of client models. For each round of communication, the server selects a subset of clients and pushes the master model to these clients. Each client then performs a predetermined number of rounds of gradient descent using the client's local data, pushes their model weight deltas to the server, and the server then averages these updates to become the new master model.

FedAvg features a fixed learning rate across devices as per normal minibatch-SGD. As shown below, this can result in a very large number of rounds required to

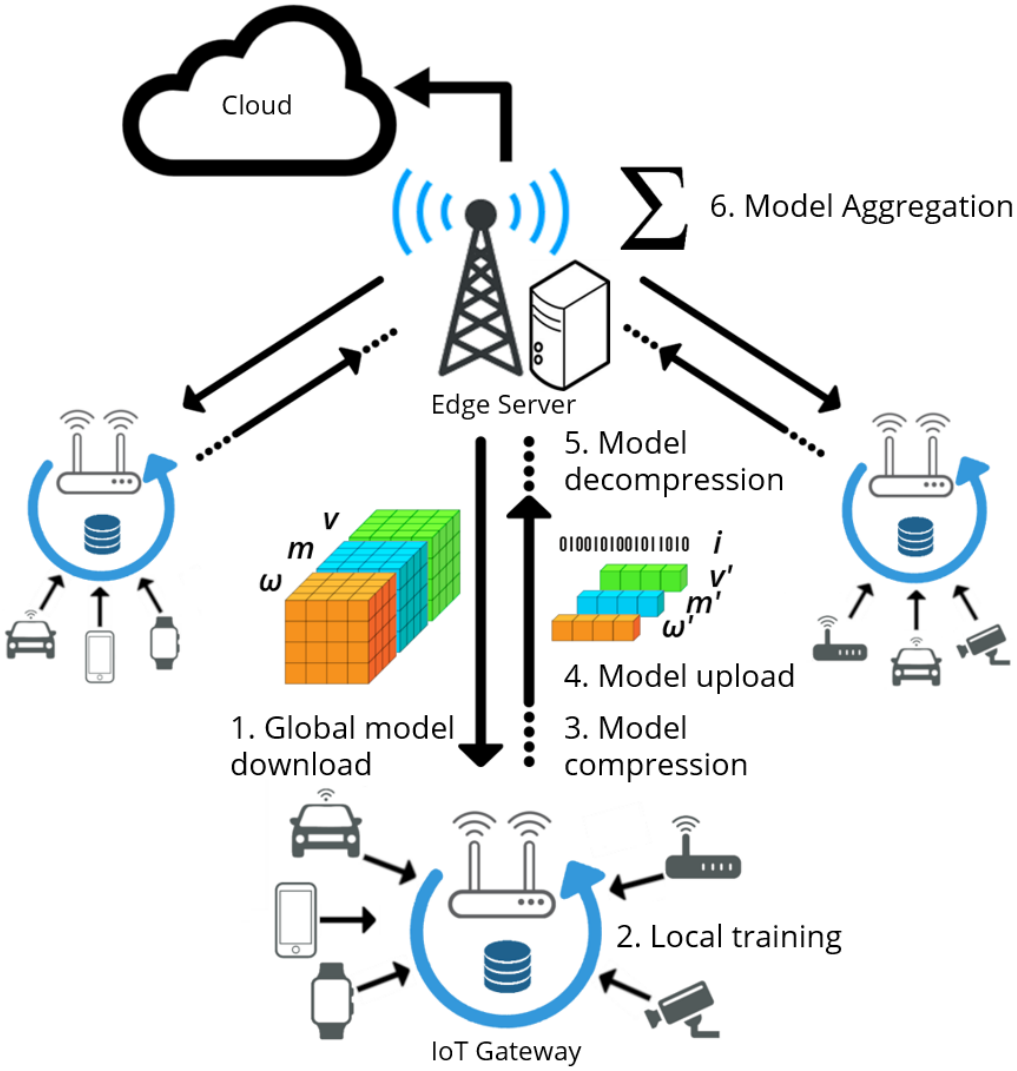


Figure 3.1: One communication round of CE-FedAvg: 1. Clients download the global model weights, 1st and 2nd moments (ω, m, v); 2. Clients perform training on their IoT-derived datasets; 3. Clients compress their models; 4. Clients upload their compressed parameters and indexes (ω', m', v', i); 5. The server decompresses the model weights and moments; 6. The server aggregates all client models before starting a new round of training.

converge to a given accuracy. minibatch-SGD can result in low convergence rates in later rounds of training because some weights need finer 'tuning' than others. Adam optimization is a popular addition to standard minibatch-SGD [87]. It stores two values for each model weight: m (the 1st moment) and v (the 2nd moment), which are used along with gradients computed by backpropagation and global decaying learning rates to update model weights for each minibatch. Adam reduces both the problems of weights requiring different degrees of tuning (having adaptive rates for each weight), and local minima (via momentum).

Alongside this, in FedAvg, the entire DNN model is sent from clients to server each round. Modern DNNs have a huge number of weights, and the upload bandwidth of edge clients is typically far lower than the download bandwidth. Therefore, uploading models to the server is a significant potential bottleneck of the system. Previous works have shown that these uploaded weight tensors can be compressed without significantly harming the performance of the model [43, 44, 89] [45]. How-

ever, these schemes typically increase the number of rounds FedAvg takes to converge, albeit with lower total data uploaded by clients.

To address the above problems, CE-FedAvg is proposed: a scheme that both reduces the number of rounds taken to converge to a given accuracy *and* decreases the total data uploaded during training over FedAvg. Algorithm 2 shows the details of CE-FedAvg. The UniQ, UniDQ, ExpQ and ExpDQ functions are shown in Algorithms 3 and 4.

Algorithm 2: Communication-Efficient FedAvg (CE-FedAvg)

server Executes:
input: initial global model $\boldsymbol{x_0}$
for round $r = 1$ **to** R **do**
  select round clients $\mathcal{C}_r$
  for client $c \in \mathcal{C}_r$ in parallel **do**
    $(\boldsymbol{\omega}_q, \mathbf{m}_q, \mathbf{v}_q, g, b^*, lz_{min}, lz_{max}, gz_{min}, gz_{max}, b_m, b_v) \leftarrow \text{ClientUpdate}_\text{k}(\boldsymbol{\omega}, \mathbf{m}, \mathbf{v})$
    decompress model and optimiser deltas
    $\Delta\boldsymbol{\omega}_k, \Delta\mathbf{m}_k, \Delta\mathbf{v}_k \leftarrow \mathbf{0}$
    $idxs \leftarrow \text{GDecode}(g, b^*)$
    $\Delta\boldsymbol{\omega}_{k,idxs} \leftarrow \text{UniDQ}(\boldsymbol{\omega}_q, lz_{min}, lz_{max}, gz_{min}, gz_{max})$
    $\Delta\mathbf{m}_{k,idxs} \leftarrow \text{ExpDQ}(\mathbf{m}_k, b_m)$
    $\Delta\mathbf{v}_{k,idxs} \leftarrow \text{ExpDQ}(\mathbf{v}_k, b_v)$
    update global values
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \sum_{k=1}^{K} \frac{n_k}{n} \Delta\boldsymbol{\omega}_k$
    $\mathbf{m} \leftarrow \mathbf{m} + \sum_{k=1}^{K} \frac{n_k}{n} \Delta\mathbf{m}_k$
    $\mathbf{v} \leftarrow \mathbf{v} + \sum_{k=1}^{K} \frac{n_k}{n} \Delta\mathbf{v}_k$
  end
end
function $\text{ClientUpdate}_\text{k}(\boldsymbol{\omega}, \mathbf{m}, \mathbf{v})$
  download values $\boldsymbol{\omega}_k \leftarrow \boldsymbol{\omega}$; $\mathbf{m}_k \leftarrow \mathbf{m}$; $\mathbf{v}_k \leftarrow \mathbf{v}$
  for epoch $\leftarrow 1$ **to** E **do**
    batches $\leftarrow$ (data P_k in batches of size B)
    for batch $b \in$ batches **do**
      $\boldsymbol{\omega}_k, \mathbf{m}_k, \mathbf{v}_k \leftarrow \text{AdamSGD}(\boldsymbol{\omega}_k, \mathbf{m}_k, \mathbf{v}_k)$
    end
  end
  compress $\boldsymbol{\omega}_k$, $\mathbf{m}_k$, $\mathbf{v}_k$ deltas and indexes
  $\boldsymbol{\omega}_s, idxs \leftarrow \text{Sparsify}(\boldsymbol{\omega}_k - \boldsymbol{\omega})$
  $g, b^* \leftarrow \text{GEncode}(idxs)$
  $\boldsymbol{\omega}_q, lz_{min}, lz_{max}, gz_{min}, gz_{max} \leftarrow \text{UniQ}(\boldsymbol{\omega}_s)$
  $\mathbf{m}_q, b_m \leftarrow \text{ExpQ}(\mathbf{m}_k - \mathbf{m})$
  $\mathbf{v}_q, b_v \leftarrow \text{ExpQ}(\mathbf{v}_k - \mathbf{v})$
  return $(\boldsymbol{\omega}_q, \mathbf{m}_q, \mathbf{v}_q, g, b^*, lz_{min}, lz_{max}, gz_{min}, gz_{max}, b_m, b_v)$ to server

In CE-FedAvg, the server first initialises the global model with random weights and 0 values for the 1st and 2nd Adam moments (line 3). Each communication round, the

server selects a subset of clients (line 5) and sends the weights and moments to them. Algorithm 2 are selects clients uniformly at random, but in reality clients would be selected based on their power/communication properties at the time as in [8]. These clients then perform their local Adam SGD (lines 7-8), and upload their data to the server. The server dequantizes and reconstructs the sparse updates from the clients (lines 10-17), averages the updates (lines 20-22) and starts the next round. Each client updates by replacing their current model with the downloaded global weights and moments (line 27), performing E epochs of Adam SGD (lines 28-34), sparsifying and then quantizing the model deltas and sparse indexes (lines 36-40) and uploading the model to the server. Barring compression, CE-FedAvg would communicate 3× the data as FedAvg (the shapes of m and v are the same as ω). However, as shown later, ω, m and v can be aggressively compressed in CE-FedAvg while still taking less rounds to converge than FedAvg.

CE-FedAvg provides other practical benefits over FedAvg. Due to the adaptive learning rates inherited from Adam, CE-FedAdam works well with the default Adam parameters. FedAvg, on the other hand, requires finding learning rates for each specific dataset and scenario [24]. Not only is this time-consuming and costly, but in the FL setting, the server does not have access to client datasets, so it is unclear how this would be done in reality. Also, as CE-FedAvg reduces the number of rounds of communication to reach a target accuracy, the total amount of computation required of clients is also reduced: the extra cost of performing one round of CE-FedAvg over FedAvg is outweighed by reduced rounds of learning.

3.2 Compression Strategies

Previous work compressing the models uploaded by clients typically rely on sparsification of weights and/or quantization of weights from typical 32-bit floats.

Algorithm 3: Uniform Quantization (UniQ) and Dequantization (UniDQ)

1 **function** UniQ($\mathbf{a}$)
2 $\quad lz_{min} \leftarrow min(\mathbf{a}^-)$
3 $\quad lz_{max} \leftarrow max(\mathbf{a}^-)$
4 $\quad gz_{min} \leftarrow min(\mathbf{a}^+)$
5 $\quad gz_{max} \leftarrow max(\mathbf{a}^+)$
6 $\quad$ initialise as type 8-bit int $\mathbf{q} \leftarrow \mathbf{0}_{|\mathbf{a}|}$
7 $\quad \mathbf{q}_{\mathbf{a}<0} \leftarrow \lfloor \frac{127}{lz_{max}-lz_{min}}(\mathbf{a}_{\mathbf{a}<0} - lz_{min}) \rfloor$
8 $\quad \mathbf{q}_{\mathbf{a}>0} \leftarrow 128 + \lfloor \frac{127}{gz_{max}-gz_{min}}(\mathbf{a}_{\mathbf{a}>0} - gz_{min}) \rfloor$
9 $\quad$ return $\mathbf{q}, \mathrm{lz_{min}}, \mathrm{lz_{max}}, \mathrm{gz_{min}}, \mathrm{gz_{max}}$
10 **function** UniDQ($\mathbf{q}, \mathrm{lz_{min}}, \mathrm{lz_{max}}, \mathrm{gz_{min}}, \mathrm{gz_{max}}$)
11 $\quad$ initialise as type 32-bit float $\mathbf{d} \leftarrow \mathbf{0}_{|\mathbf{q}|}$
12 $\quad \mathbf{d}_{\mathbf{q}<128} \leftarrow (\frac{lz_{max}-lz_{min}}{127}\mathbf{q}_{\mathbf{q}<128}) + lz_{min}$
13 $\quad \mathbf{d}_{\mathbf{q}\geq 128} \leftarrow (\frac{gz_{max}-gz_{min}}{127}(\mathbf{q}_{\mathbf{q}\geq 128} - 128)) + gz_{min}$
14 $\quad$ return $\mathbf{d}$

The proposed techniques are comprised of sparsification followed by quantization. To sparsify gradients, for each tensor, the top $(s-1)\%$ of deltas with the largest absolute value are chosen, and they are extracted along with their (flattened) indexes. In CE-FedAvg, the corresponding m and v deltas are also sent along with the

weight deltas, using the same indexes. If non-corresponding m and v values are sent (i.e. the m and v tensors are sparsified independently, in the same manner as the weight tensors) this quickly produces exploding gradients at the clients in practice, likely due to stale m and v values producing problems in the Adam optimization algorithm.

After sparsification, the weight deltas, m and v are quantized from 32-bit floats to 8-bit unsigned integers. The weight deltas contain positive and negative values with the greatest $(s-1)\%$ of magnitudes, and are quantized as per the *Uniform Quantization* scheme shown in Algorithm 3. To quantize using Uniform Quantization, the values with the greatest positive and greatest negative, and lowest positive and lowest negative are chosen (lines 2-5). These are used to map all values lower than zero to the integers 0-127 (line 7) and values greater than zero to 128-255 (line 9). To dequantize, the reverse functions are applied (lines 14-15) to return a vector of floats.

Analysis of m and v values produced by Adam show that there is a large range in the scale of these values: usually values with exponents ranging from 10^{-1} to 10^{-35}. Attempts to quantize m and v deltas show that they are very sensitive to errors in their exponent, and quantizing them using Uniform Quantization or schemes from other works results in exploding gradients within a few rounds. Therefore a different method is proposed for these deltas: *Exponential Quantization*.

Algorithm 4: Exponential Quantization (ExpQ) and Dequantization (ExpDQ)

1 **function** ExpQ($\mathbf{a}$)
2 $\quad b \leftarrow min(abs(\mathbf{a}))^{\frac{1}{127}}$
3 $\quad$ initialise as type 8-bit int $\mathbf{q} \leftarrow \mathbf{0}_{|a|}$
4 $\quad \mathbf{p} \leftarrow \lfloor \frac{1}{log(b)} log(abs(\mathbf{a})) \rfloor$
5 $\quad \mathbf{q}_{\mathbf{a}<0} \leftarrow abs(\mathbf{p}_{\mathbf{a}<0})$
6 $\quad \mathbf{q}_{\mathbf{a}>0} \leftarrow 128 + abs(\mathbf{p}_{\mathbf{a}>0})$
7 $\quad$ return $\mathbf{q}$, b
8 **function** ExpQ($\mathbf{q}$, b)
9 $\quad$ initialise as type 32-bit float $\mathbf{d} \leftarrow \mathbf{0}_{|\mathbf{q}|}$
10 $\quad \mathbf{d}_{0<\mathbf{q}<128} \leftarrow -b^{-\mathbf{q}_{0<\mathbf{q}<128}}$
11 $\quad \mathbf{d}_{\mathbf{q}\geq 128} \leftarrow b^{128-\mathbf{q}_{\mathbf{q}\geq 128}}$
12 $\quad$ return $\mathbf{d}$

In Exponential Quantization, negative values are again mapped to the range 0-127, and positive to 128-255. Algorithm 4 shows the procedure. To quantize a tensor, the smallest absolute value, δ, is found, and $b = \delta^{-1/127}$ is computed (line 2). This provides the *base* for quantization: the minimum base with exponent of 127 able to represent the value with the smallest magnitude in the tensor. Using the minimum possible base provides the highest resolution for the quantized values. The logarithm with base b for all values in the tensor is then found and rounded to the nearest integer (line 4), with 128 added to positive values to map them to 128-255 (line 6). To dequantize, b is simply raised to the power of $\mathbf{q}$ for each value, times by -1 for $\mathbf{q} < 128$ (lines 12-13).

The last item to be compressed are the indexes of the values in the sparse arrays sent from clients to server. For these values, lossless Golomb Encoding is used [88].

Using these techniques, compressed FedAvg therefore uploads for each weight tensor: the weight deltas (8-bit integers); the four min/max values from Uniform

Quantization (4×32 bits); their indexes (Golomb encoded); and b^* (a 32-bit float used for Golomb encoding). CE-FedAvg must communicate all of these, plus the m and v deltas (both 8-bit integers) and the base, b for both of these from Exponential Quantization (32-bit floats). The total number of uploaded bits after compression, per client per round, is therefore:

$$\text{FedAvg: } bits_{up} = 160|W| + (1-s)(8+g_s)\sum_{\boldsymbol{\omega}\in\Omega}|\boldsymbol{\omega}| \tag{3.1}$$

$$\text{CE-FedAvg: } bits_{up} = 224|W| + (1-s)(24+g_s)\sum_{\boldsymbol{\omega}\in\Omega}|\boldsymbol{\omega}| \tag{3.2}$$

where s is the sparsity, Ω is the set of weight tensors comprising the network, and g_s is the expected number of bits needed to Golomb encode one index value for a given sparsity.

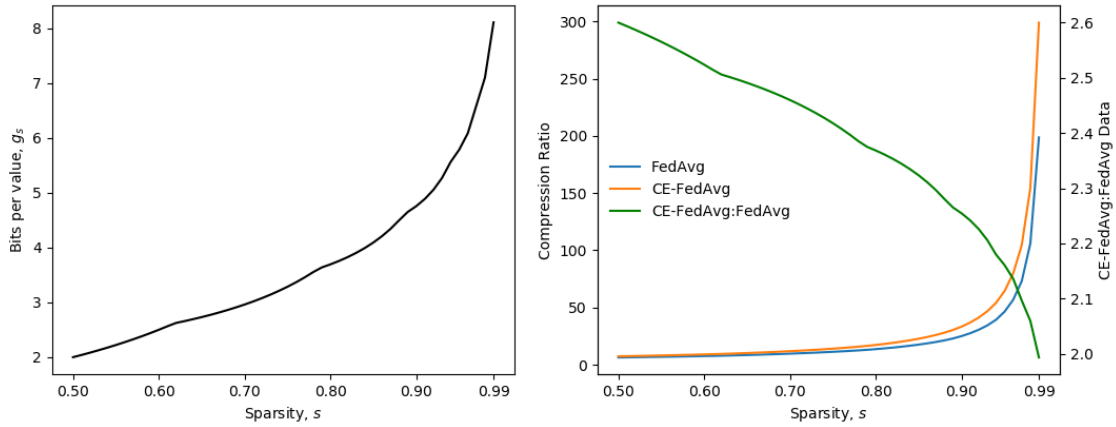


Figure 3.2: *Left:* expected number of bits per value, g_s, using Golomb Encoding versus sparsity. *Right:* compression ratio of CE-FedAvg and FedAvg, and CE-FedAvg:FedAvg uploaded data ratio versus sparsity.

The implementation of Golomb Encoding (GE) [88] is the same as used in [45]. The expected number of bits per value for a given sparsity, g_s, is given as:

$$g_s = b^* + \frac{1}{1 - s^{2^{b^*}}} \tag{3.3}$$

$$b^* = 1 + \lfloor \log_2(\frac{\phi - 1}{s}) \rfloor \tag{3.4}$$

where $\phi = \frac{\sqrt{5}+1}{2} \approx 1.62$ is the golden ratio. The value of b^* is sent from client to server to convert the GE bit string back to integer values.

Plotting g_s (Figure 3.2, left), the second terms of the above equations, and the nominal (without $|W|$ terms) ratio of CE-FedAvg to FedAvg uploaded data with different s (Figure 3.2, right) shows that CE-FedAvg compresses more than FedAvg using this scheme. This means the total amount of data uploaded by a CE-FedAvg client in a single round is between $2 - 2\times$ that of FedAvg for $0.5 \leq s < 1.0$, as opposed to $3\times$ if there was no compression. However, despite communicating $2 - 2.6\times$ more data than FedAvg per client per round, due to the reduction in rounds achieved by CE-FedAvg, clients still upload less total data than FedAvg in many cases.

3.3 Experimental Evaluation

In this section, simulated and testbed experiments are performed to compare the performance of CE-FedAvg against FedAvg in terms of the number of rounds to reach a target average test accuracy (communication-efficiency), total communicated data, and runtime.

3.3.1 Simulation Setup

Simulated experiments were conducted on image classification tasks using the MNIST [90] and CIFAR10 [91] datasets to demonstrate the benefit of CE-FedAvg in terms of communication-efficiency. DNN models were implemented using Tensorflow [92].

> *MNIST*: 28×28 greyscale images of handwritten digits in 10 classes. Two models were trained on this dataset. The first (MNIST-2NN) had two fully connected layers of 200 neurons with ReLU activation, and a softmax output layer. The second (MNIST-CNN) was a convolutional network consisting of: two 5×5 convolutional layers with 32 and 64 output neurons, respectively, each followed by 2×2 max pooling and ReLU activation; a fully connected layer with 512 neurons and ReLU activation; and a softmax output layer.
>
> *CIFAR10*: 32×32 colour images of objects in 10 classes. One network (CIFAR-CNN) was trained on this dataset with: two 3×3 convolutional layers with 32 output neurons, L2 regularization and each followed by batch normalization; a dropout layer with $d = 0.2$, two 3×3 convolutional layers with 64 output neurons, L2 regularization, and batch normalization; a second dropout layer with $d = 0.3$; two final 3×3 convolutional layers with L2 regularization and batch normalization; a final $d = 0.4$ dropout layer; and a softmax output layer.

The networks were trained on the datasets using differing numbers of clients, classes per client, client participation rate, compression rates, and either FedAvg or CE-FedAvg until a given target accuracy was achieved. The models had the following target accuracies: MNIST-2NN, 97%; MNIST-CNN, 99%; CIFAR-CNN, 60%. For each setting, values of E were tested to find the best for that setting, and when using FedAvg, different learning rates were also tested.

The entire dataset was split across all clients in each case. Therefore, increasing numbers of clients resulted in less samples per worker. To produce IID data ($Y = 10$), the datasets were shuffled and each client given an equal portion of the data. For non-IID data ($Y = 2$), the datasets was sorted by class, divided into slices, and each client was either given two slices. This resulted in most clients having data from only two classes. The test data for each dataset was taken from the official test-sets.

3.3.2 Simulation Results

The following tables show the average number of rounds to reach a given target accuracy for the best parameter setup in each case. 'NC' is used where no set of parameters could be found to get the algorithm to converge to the target. Sparsity rates of s $= \{0.6, 0.9, 0.95\}$ provides approximately $\{7, 25, 53\}\times$ compression for

FedAvg, and approximately $\{9, 33, 68\}\times$ compression for CE-FedAvg, respectively, resulting in a FedAvg:CE-FedAvg uploaded data ratio per round of $\approx 1 : 2.3$. For FedAvg, multiple global learning rates were also trialled for each scenario.

Table 3.1: Rounds required to reach target test accuracies for FedAvg (grey) and CE-FedAvg (white), with upload sparsity $s = 0.6$.

MNIST-2NN						
Y	$W = 10$		20		40	
	$C = 0.5$	1.0	0.5	1.0	0.5	1.0
10	2.0	2.0	3.8	4.0	7.2	7.2
	2.8	2.0	4.2	4.0	8.2	7.4
2	134.8	118.0	143.8	117.2	171.6	152.4
	79.4	56.2	77.6	60.8	81.6	60.8
MNIST-CNN						
10	3.2	3.0	7.2	4.4	10.0	9.6
	3.0	3.0	4.2	4.0	8.6	8.4
2	143.2	121.4	146.6	119.4	190.2	230.0
	56.4	36.2	54.0	41.0	58.8	43.0
CIFAR-CNN						
10	3.0	3.0	5.2	5.0	10.6	9.8
	2.8	3.0	3.0	3.0	5.2	5.4
2	111.2	99.2	159.2	138.2	225.2	241.0
	80.2	82.8	58.6	75.4	53.8	92.7

Table I shows that more rounds are required to converge in non-IID scenarios than IID, as is consistent with other FL works. As clients' distributions are very different in non-IID scenarios, their models diverge more between aggregations, harming the global model. Table I also shows that with moderate compression, CE-FedAvg was able to reach the target in significantly fewer rounds in non-IID scenarios. This may be because of CE-FedAvg's adaptive learning rates: Adam was able to make more fine-tuned changes to the worker models between aggregations so these models did not diverge as much. CE-FedAvg did comparatively better with more workers/decreasing dataset size per worker. CE-FedAvg reduced the rounds taken by $\geq 2.3\times$ in 10 cases, including all the MNIST-CNN non-IID cases. In the MNIST-CNN, $W = 40$, $C = 0.5$, $Y = 2$ case, CE-FedAvg reduced the number of rounds by $5.3\times$ over FedAvg.

Tables II and III show that higher compression rates increase the number of rounds to reach the target, especially in non-IID cases. Again, in all non-IID cases, CE-FedAvg is able to reach the target far faster than FedAvg. For $s = 0.9$, CE-FedAvg reached the target in $\geq 2.3\times$ less rounds in 7 cases. For $s = 0.95$, CE-FedAvg took $\geq 2.3\times$ less rounds in 5 cases. Taking the same MNIST-CNN case, with $s = 0.9$, CE-FedAvg reaches the target in $6\times$ fewer rounds, and $4.3\times$ fewer for $s = 0.95$.

For all non-IID CIFAR-10 $s = 0.9, 0.95$ cases, compressed FedAvg could not converge within 1000 rounds. The CIFAR-10 problem is much more complex than MNIST. It is likely FedAvg could not converge due to its fixed learning rate. CE-FedAvg, on the other hand, could reliably converge even in these extreme settings.

Table 3.2: Rounds required to reach target test accuracies for FedAvg (grey) and CE-FedAvg (white), with upload sparsity $s = 0.9$. 'NC' denotes cases unable to converge.

Y	$W = 10$		20		40	
	$C = 0.5$	1.0	0.5	1.0	0.5	1.0
MNIST-2NN						
10	3.0	2.8	4.8	4.8	9.6	9.6
	4.2	4.0	7.0	6.0	13.0	11.6
2	210.2	192.0	212.2	185.6	271.8	258.4
	114.8	86.4	104.6	86.4	138.4	96.8
MNIST-CNN						
10	4.4	3.8	7.4	6.4	14.2	13.4
	4.8	4.8	7.8	7.4	13.6	12.4
2	256.8	184.4	216.6	462.8	495.6	447.6
	85.8	50.2	78.0	56.0	79.2	74.8
CIFAR-CNN						
10	4.4	4.0	8.2	7.0	14.0	14.2
	5.2	5.6	6.0	6.0	9.2	9.4
2	NC	NC	NC	NC	NC	NC
	83.2	60.4	74.2	68.2	75.8	71.4

Not considering these cases, CE-FedAvg was less likely to diverge during training than FedAvg. Over the total 962 FedAvg and 1034 CE-FedAvg experiments conducted (after finding suitable E values, and learning rates for FedAvg, and not including the 'NC' FedAvg cases), FedAvg diverged before reaching the target in 2% of the experiments, whereas CE-FedAvg diverged in 0.1% of cases (a single case). This reliability is likely due to Adam's adaptive updates: discrepancies between the model weights downloaded from the server and what is suitable for the specific client's optimisation are more easily overcome with adaptive learning rates.

While also being more reliable than FedAvg, CE-FedAvg was able to achieve this using the default parameters for Adam optimization in every case. This resulted in much faster experiment set times for CE-FedAvg, as multiple learning rates did not have to be trialled. Tuning the Adam parameters may have achieved even better results than those listed above. FL considers machine learning where a central server does not have access to training data due to client privacy. Therefore, this presents a major advantage over FedAvg: without a central test/validation set of data, it would be infeasible to test multiple learning rates for FedAvg before conducting the actual FL, whereas CE-FedAdam, for all of the above experiments, worked out of the box with no parameter tuning.

Figure 3 shows the compressed size of the MNIST-CNN updates for FedAvg and CE-FedAvg, including an uncompressed case ($s = 0$). It is interesting to see that while CE-FedAvg uploads more data per client per round (due to the extra variables from Adam) in all cases, the total data uploaded by CE-FedAvg is far lower than FedAvg in all cases. This is due to the large decrease in rounds to convergence CE-FedAvg gives.

Table 3.3: Rounds required to reach target test accuracies for FedAvg (grey) and CE-FedAvg (white), with upload sparsity $s = 0.95$. 'NC' denotes cases unable to converge.

Y	$W = 10$		20		40	
	$C = 0.5$	1.0	0.5	1.0	0.5	1.0
MNIST-2NN						
10	3.8	3.3	6.3	6.0	13.0	12.5
	6.0	6.0	10.0	9.8	19.3	17.8
2	397.0	290.4	338.0	293.2	308.8	362.8
	173.8	130.6	156.0	139.2	211.4	169.2
MNIST-CNN						
10	4.6	4.6	9.6	9.0	21.0	19.4
	7.8	7.0	13.0	11.5	21.0	19.0
2	264.2	282.3	400.3	421.5	585.3	698.3
	133.8	94.3	126.3	107.8	160.8	161.8
CIFAR-CNN						
10	5.6	5.0	9.6	8.6	20.2	18.0
	10.0	10.6	10.2	9.6	14.2	13.8
2	NC	NC	NC	NC	NC	NC
	173.6	126.6	148.6	112.4	306.0	186.8

3.3.3 Testbed Setup

To test the real-time convergence of CE-FedAvg over FedAvg, an Raspberry-Pi (RPi) testbed was used to simulate a heterogeneous low-powered edge-computing scenario. The testbed consisted of 5 Raspberry Pi 2Bs and 5 Raspberry Pi 3Bs. A desktop acted as the server over a wireless network to emulate lower-bandwidth networking. The work of the server in these experiments was small: receiving and decompressing, aggregating and resending models to clients. Therefore, the server had a small impact on the time experiments took to run, and the vast majority of time taken in the FedAvg/CE-FedAvg algorithm was on the RPi clients. The RPi clients ran Raspbian OS and software was written with Python using Tensorflow 1.12.

Experiments with 10 workers, the MNIST-2NN and MNIST-CNN models, and sparsity rate $s = 0.6$ were performed to evaluate the runtime of CE-FedAvg and FedAvg. Round times were taken and averaged, and then used with the total-rounds from the relevant parts of Table I to determine the total time that experiment would take on the testbed. Time taken to complete rounds was very consistent on the testbed, making this a reliable estimator of total time.

3.3.4 Testbed Results

Experiments were performed to obtain the estimated real time on a RPi testbed for a set of 4 experiments to reach given target accuracies. The results of this are shown in Figure 3.4.

The times in Figure 4 show that CE-FedAvg is able to converge to a given target accuracy in less real time than FedAvg with similar compression. Although the

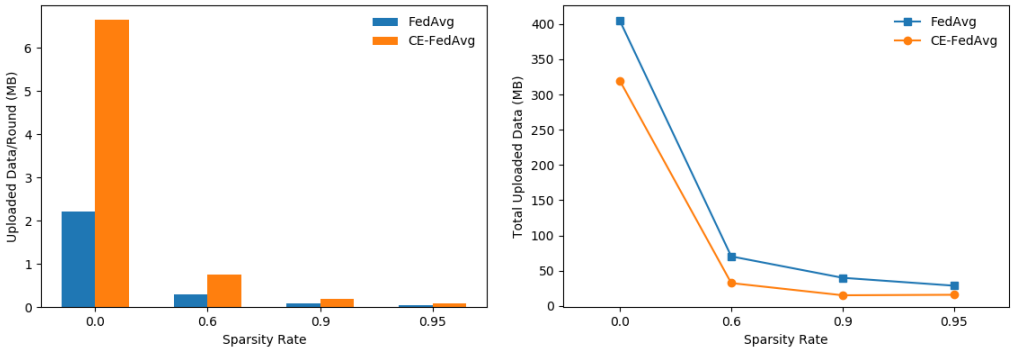


Figure 3.3: *Left:* Compressed uploaded data per client per round for FedAvg and CE-FedAvg with different sparsities, for the MNIST-CNN $W = 40$, $C = 1.0$, $Y = 2$ scenario. *Right:* Total uploaded data during training for the same scenario.

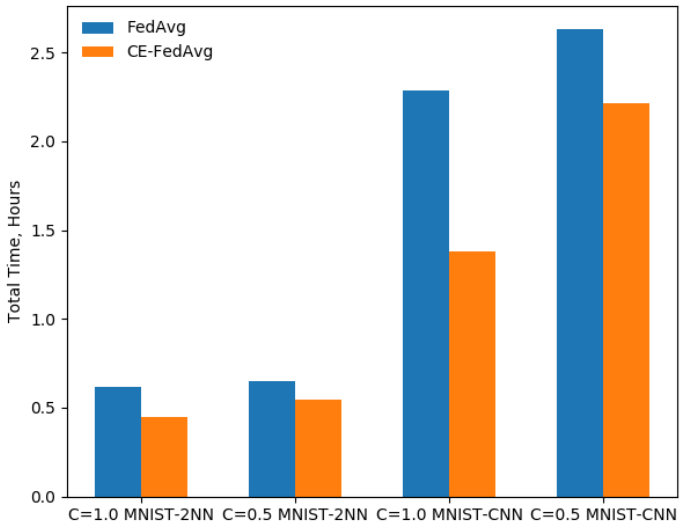


Figure 3.4: Estimated time on Raspberry Pi testbed of different FL scenarios.

time taken per round is greater for CE-FedAvg than FedAvg (due in small extra computation required from Adam, but mostly due to the increased communication per round compared to similarly compressed FedAvg), the number of rounds taken to converge, as per Table I, was lower in all the given experiments. Figure 4 shows CE-FedAvg was able to converge $1.2 - 1.7\times$ faster than FedAvg.

3.4 Chapter Summary

Federated Learning (FL) can allow distributed Machine Learning to be performed on the network edge using data generated by IoT devices. In this chapter, Adam optimisation and novel compression techniques were added to FedAvg to produce Communication-Efficient FedAvg (CE-FedAvg), which reduces the total uploaded data and rounds required to reach a given model accuracy, compared to similarly-compressed FedAvg. Extensive experiments on the MNIST and CIFAR-10 datasets showed CE-FedAvg was generally able to reach a target accuracy in far fewer communication rounds than FedAvg in non-IID settings (up to $6\times$ fewer). These experiments showed CE-FedAvg is also far more robust to aggressive compression of uploaded data, and able to converge with up to $3\times$ less total uploaded data per client. Further experiments using a Raspberry-Pi testbed showed CE-FedAdam

could converge in up to 1.7× less real-time. CE-FedAvg therefore presents the benefits of being able to train a model in less communication rounds (reducing the overall data and computing cost of training), less real-time, and with less uploaded data than uncompressed FedAvg, a unique result considering most schemes using compression reduce uploaded data at the cost of more total communication rounds. Future work in this area could investigate other SGD-type algorithms applied to FL, and in compressing the models downloaded by clients from the server.

Chapter 4

Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing

Previous chapters have shown that non-Independent and Identically Distributed (non-IID) user data is central to the FL scenario, but harms the convergence speed of FedAvg and related algorithms [14]. Furthermore, most existing FL works measure the performance of the trained global model, but in many cases such as user content-recommendation, improving individual User model Accuracy (UA) on their private data is the real objective. When using a single global model, UA will be non-uniform across clients due to their non-IID data. To address these issues, this chapter proposes a Multi-Task FL (MTFL) algorithm that introduces non-federated Batch-Normalization (BN) layers into the federated DNN. These weights in these layers are not averaged with the rest of the model parameters but kept private. MTFL benefits UA and convergence speed by allowing users to train models personalised to their own data, and is compatible with iterative FL algorithms such as FedAvg. This chapter also shows empirically that a distributed form of Adam [87] optimisation (FedAvg-Adam) benefits convergence speed even further when used as the optimisation strategy within MTFL. Experiments using MNIST and CIFAR10 demonstrate that MTFL is able to significantly reduce the number of rounds required to reach a target UA, by up to 5× when using existing FL optimisation strategies, and with a further 3× improvement when using FedAvg-Adam. MTFL is compared with existing personalised-FL algorithms, showing that it is able to achieve the best UA for MNIST and CIFAR10 in all considered scenarios. Finally, MTFL with FedAvg-Adam is evaluated on an edge-computing testbed, showing that its convergence and UA benefits outweigh its overhead.

4.1 Multi-Task Federated Learning (MTFL)

Figure 4.1 shows a high-level overview of MTFL operation in the edge-computing environment. More detailed descriptions of the use of BN patches in MTFL and optimisation on clients is given in the later subsections.

The MTFL algorithm is based on the client-server framework, however rounds are initiated by the server, as shown in Figure 4.1. First, the server selects all, or a subset of all, known clients from its database and asks them to participate in the FL round (**Step 1**), and sends a *Work Request* message to them. Clients will accept a *Work Request* depending on user preferences (for example, users can set their device to only participate in FL if charging and connected to Wi-Fi). All accepting clients

then send an *Accept* message to the server (**Step 2**). The server sends the global model (and any associated optimization parameters) to all accepting clients, who augment their copy of the global model with private patches (**Step 3**). Clients then perform local training using their own data, creating a different model. Clients save the patch layers from their new model locally, and upload their non-private model parameters to the server (**Step 4**).

The server waits for clients to finish training and upload their models (**Step 5**). It can either wait for a maximum time limit, or for a given fraction of clients to upload before continuing, depending on the server preferences. After this, the server will aggregate all received models to produce a single global model (**Step 6**) which is saved on the server, before starting a new round.

MTFL therefore offloads the vast majority of computation to client devices, who perform the actual model training. It preserves users' data-privacy more strongly than FedAvg and other personalised-FL algorithms: not only is user data not uploaded, but key parts of their local models are not uploaded. The framework also accounts for client stragglers with its round time/uploading client fraction limit. Moreover, MTFL utilises patch layers to improve *local* model performance on individual users' non-IID datasets, making MTFL more personalised.

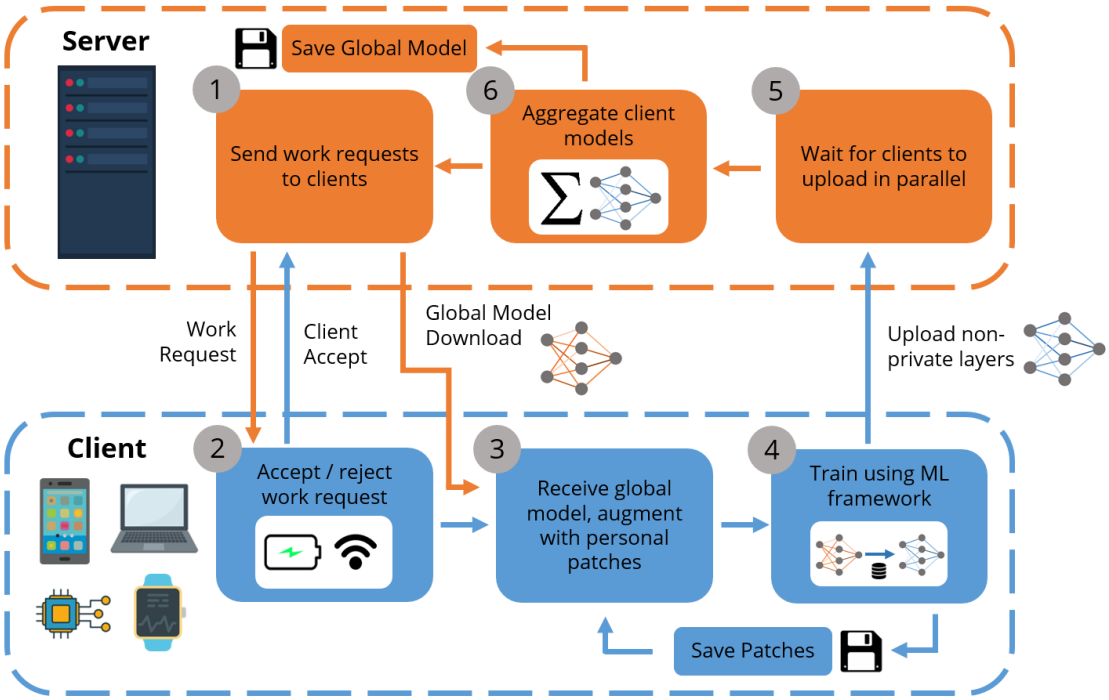


Figure 4.1: Operation of the MTFL algorithm in Edge Computing. Training is performed in rounds until a termination condition is met. **Step 1**: the server selects a subset of clients from its database to participate in the round, and sends a work request to them. **Step 2**: clients reply with an accept message depending on physical state and local preferences. **Step 3**: clients download the global model (and any optimisation parameters) from the server, and update their copy of the global model with private patches (in this work, we use BN layers as patches). **Step 4**: clients perform local training, before saving their personal patches for the next round. **Step 5**: the server waits for C fraction of clients to upload their non-private model and optimiser values, or until a time limit. **Step 6**: the server averages all models, saves the aggregate, and starts a new round.

4.1.1 User Model Accuracy and MTFL

In many FL works, such as the original FedAvg paper [24], the authors use a central IID test-set to measure FL performance. Depending on the FL scenario, this metric may or may not be desirable. If the intention is to create a single model that has good performance on IID data, then this method would be suitable. However, in many FL scenarios, the desire is to create a model that has good performance on individual user devices. For example, Google have used FedAvg for their GBoard next-word-prediction software [8]. The objective was to improve the prediction score for individual users. As users do not typically have non-IID data, a single global model may display good performance for some users, and worse performance for others.

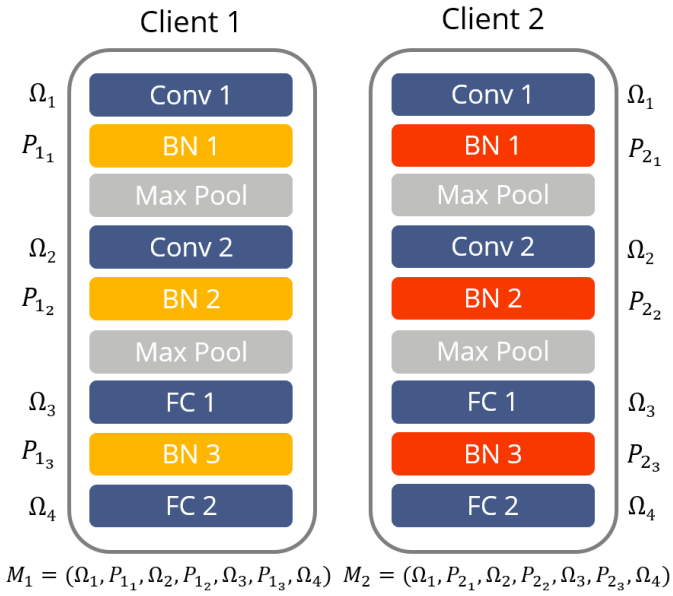


Figure 4.2: Example composition of a DNN model used in MTFL. Each client's model consists of shared global parameters ($\Omega_1 - \Omega_4$) for Convolutional (Conv) and Fully-Connected (FC) layers, and private Batch-Normalization (BN) patch layers ($P_{k_1}, P_{k_2}, P_{k_3}$).

Instead, User model Accuracy (UA) can be used as an alternative metric of FL performance. UA is the accuracy on a client using a local test-set. This test-set for each client should be drawn from a similar distribution as its training data. In this chapter experiments are performed on classification problems, but UA could be altered for different metrics (e.g. error, recall).

In FL, user data is often non-IID, so users could be considered as having different but related learning tasks. It is possible for an FL scheme to achieve good global-model accuracy, but poor UA, as the aggregate model may perform poorly on some clients' datasets (especially if they have a small number of local samples, so are weighted less in the FedAvg averaging step). The MTFL algorithm that allows clients to build different models while still benefiting from FL, in order to improve the average UA. Mudrakarta *et al.* [93] previously shown that adding small per-task 'patch' layers to DNNs improved their performance in MTL scenarios. Patches are therefore a good candidate for training personalised models for clients.

In FL, the aim is to minimise the following objective function:

$$F_{\text{FL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\Omega), \tag{4.1}$$

where K is the total number of clients, n_k is the number of samples on client k, n is the total number of samples across all clients, ℓ_k is the loss function on client k, and Ω is the set of global model parameters. Adding unique client patches to the FL model changes the objective function of MTFL to:

$$F_{\text{MTFL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ \ell_k(\mathcal{M}_k), \tag{4.2}$$

$$\mathcal{M}_k = (\Omega_1 \cdots \Omega_{i_1}, P_{k_1}, \Omega_{i_1+1} \cdots \Omega_{i_m}, P_{k_m}, \Omega_{i_m+1} \cdots \Omega_j), \tag{4.3}$$

where $\mathcal{M}_k$ is the patched model on client k, composed of Federated model parameters $\Omega_1 \cdots \Omega_j$ (j being the total number of Federated layers) and patch parameters $P_{k_1} \cdots P_{k_m}$ (m being the total number of local patches, $\{i\}$ being the set of indexes of the patch parameters) unique to client k. Fig. 2 shows an example composition of a DNN model used in MTFL.

MTFL is a general algorithm for incorporating MTL into FL. Different optimisation strategies (including FedAvg-Adam as described in Section 4.1.3) can be used within MTFL, and this chapter later shows that MTFL can substantially reduce the number of rounds to reach a target UA regardless of the optimisation strategy used.

Algorithm 5: Multi-Task Federated Learning (MTFL)

1 **input:** initial global model Ω and optimiser V, patchIdxs
2 **for** round $r = 1$ **to** R **do**
3 select round clients $\mathcal{C}_r$
4 **for** client $c \in \mathcal{C}_r$ in parallel **do**
5 download global model $\mathcal{M}_k \leftarrow \Omega$
6 download optimiser values $V_K \leftarrow V$
7 **for** $i \in$ patchIdxs **do**
8 apply local patch $\mathcal{M}_{k,i} \leftarrow P_{k,i}$, $V_{k,i} \leftarrow W_{k,i}$
9 **end**
10 **for** batch b drawn from local data D_k **do**
11 $\mathcal{M}_k, V_k \leftarrow \text{LocalUpdate}(\mathcal{M}_k, V_k, b)$
12 **end**
13 **for** $i \in$ patchIdxs **do**
14 save local patch $P_{k,i} \leftarrow \mathcal{M}_{k,i}, W_{k,i} \leftarrow V_{k,i}$
15 **end**
16 **for** $i \notin$ patchIdxs **do**
17 upload $\mathcal{M}_{k,i}, V_{k,i}$ to server
18 **end**
19 **end**
20 **for** $i \notin$ patchIdxs **do**
21 $\Omega_i \leftarrow \text{GlobalModelUpdate}(\Omega_i, \{\mathcal{M}_{k,i}\}_{k \in S_r})$
22 $V_i \leftarrow \text{GlobalOptimUpdate}(V_i, \{V_{k,i}\}_{k \in S_r})$
23 **end**
24 **end**

As shown in Algorithm 5, MTFL runs rounds of communication until a given termination criteria (such as number of rounds R or target UA) is met (Line 2). At each

round, a subset C_r of clients are selected to participate from the set of all clients (Line 3). These clients download the global model Ω, which is a tuple of model parameters, and the global optimiser V, if used (Lines 5-6). The clients then update their copy of the global model and optimiser with their private patch layers (Lines 7-9), where the 'patchIdxs' variable contains the indexes of patch layer placement in the DNN. Clients perform training using their now-personalised copy of the global model and optimiser on their local data (Line 10). Depending on the choice of optimisation strategy to be used within MTFL, the LocalUpdate function represents local training of the model. For FedAvg, LocalUpdate is simply minibatch-SGD. We discuss this, and the proposed FedAvg-Adam optimisation strategy, in Section 4.1.3. After local training, the updated client patches are saved (Lines 11-13), and the non-patch layers and optimiser values are uploaded to the server (Lines 14-16).

At the end of the round, the server makes a new global model and optimiser according to the GlobalModelUpdate and GlobalOptimUpdate functions (Lines 18-20). These functions are again dependent on the optimisation strategy used, and are discussed further in Section 4.1.3. FedAvg for example uses a weighted average of client models for GlobalModelUpdate. The updated global model marks the end of the round and a new round is begun.

The total per-round computation complexity of MTFL scales with $\mathcal{O}(|\mathcal{C}_r|)$, where $\mathcal{O}(|\mathcal{C}_r|)$ is the number of clients participating per round. The computation performed by each client is independent of the total number of clients. As clients perform local computation in parallel, MTFL (like FedAvg) is eminently scalable. Scalability is important in FL as real-world deployments are expected to have huge numbers of low-powered clients [94, 8]. The global model and optimiser updates (Lines 20-23 in Algorithm 1) depend on the optimisation strategy used. For FedAvg and FedAvg-Adam, GlobalModelUpdate is essentially map-reduce (averaging after local training) - also $\mathcal{O}(|\mathcal{C}_r|)$. For FedAdam, the Adam step following the map-reduce in GlobalOptimUpdate is not dependent on the number of clients (only on the DNN architecture).

There are numerous works investigating FL in the Peer-To-Peer (p2p) setting [95], which is beyond the scope of this chapter. Simple p2p FL algorithms involve sending all client models to all participating peers for decentralised aggregation. Extension of MTFL to these schemes is trivial: peers would simply just send/aggregate the non-private layers. More sophisticated p2p FL algorithms may require more complex ways of incorporating private layers – an interesting direction we leave for future works.

Mudrakarta *et al.* [93] showed that Batch Normalisation (BN) layers [96] can act as model patches for MTL in the centralised setting. We show later that BN layers work well as patches in MTFL, considering that they are very lightweight in terms of number of parameters. BN layers are given by:

$$\begin{aligned} \hat{x}_i &= \frac{z_i - \mathbb{E}(z_i)}{\sqrt{\mathrm{Var}(z_i) + \epsilon}}, \\ \mathrm{BN}(\hat{x}_i) &= \gamma_i \hat{x}_i + \beta_i, \end{aligned} \tag{4.4}$$

where $\mathbb{E}(z_i)$ and $\mathrm{Var}(z_i)$ are the mean and variance of a neuron's activations (z_i, post-nonlinearity) across a minibatch, and γ_i and β_i are parameters learned during training. BN layers track a weighted moving average of $\mathbb{E}(z_i)$ and $\mathrm{Var}(z_i)$ during training: μ_i and σ_i^2, for use at inference time. Section 4.2.2 investigates the benefit of keeping statistics μ, σ and/or trainable parameters γ, β as part of private patch layers.

BN layers are suitable for personalisation within MTFL because: 1) they show excellent personalisation performance and 2) the storage cost of BN parameters is very small ($< 1\%$ of total model size for the tested model architectures). Mudrakarta *et al.* [93] also investigated using depthwise-convolutional patches for centralised MTL. Any model layers could in principle be kept private during MTFL, however, there is an inherent trade-off between the number of parameters kept private and the ability of the global model to converge.

4.1.2 Effect of BN Patches on Inference

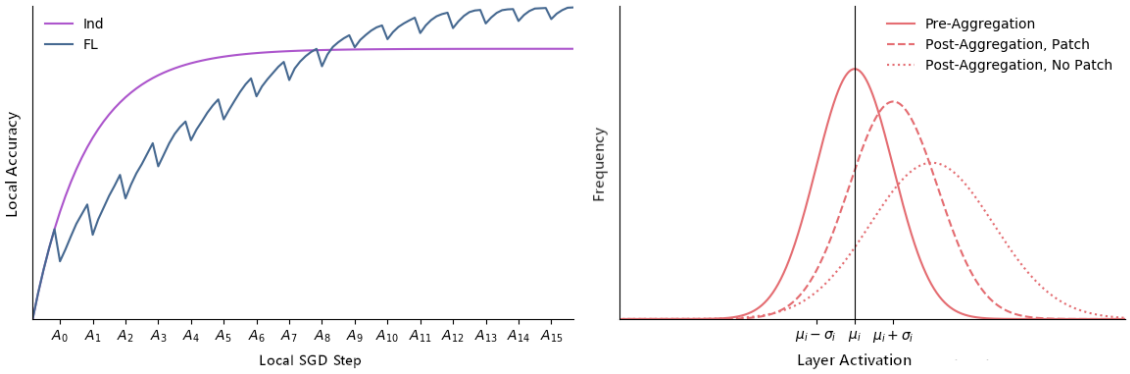


Figure 4.3: *Left:* Federated Learning (FL) results in an accuracy curve where the UA decreases after aggregations and increases during local training, compared with the smoother accuracy curve when training independently (Ind). *Right:* Patch BN-layers help bring the distribution in outputs for neuron i closer to the pre-aggregation distributions.

To understand the impact that BN-patch layers have on UA, the change in internal DNN activations over a client's local test-set immediately *before* and immediately *after* the FL aggregation step is considered.

As illustrated in Figure 4.3 (left), UA typically drops after the aggregation step in during FedAvg. This is because the model has been tuned on the local training set for several epochs, and suddenly has its model weights replaced by the Federated weights, which are unlikely to have better test performance than the pre-aggregation model. This idea is further examined in [56] and demonstrated later experimentally in Section 4.2.3. Consider a simple DNN consisting of dense layers followed by BN and then nonlinearities. The vector of first-layer neuron activations over the client's test-set (X) from applying weights and biases (W_0, b_0), can be modelled as a normal distribution:

$$\begin{aligned} z_i &\triangleq [W_0 X + b_0]_i, \\ z_i &\sim N(\mathbb{E}[z_i], Var[z_i]), \end{aligned} \tag{4.5}$$

During local training, the client's model has been adapted to the local dataset, and the BN-layer statistics used for inference (μ, σ^2) have been updated from the layer activations. Assuming, after local training (and before aggregation), $\mu_i \approx \mathbb{E}[z_i]$, and $\sigma_i^2 \approx Var[z_i]$, then the BN-layer (ignoring ϵ) computes:

$$\begin{aligned} \hat{x}_i &\triangleq \frac{z_i - \mu_i}{\sigma_i}, \\ \hat{x}_i &\sim N(0,1), \\ \mathrm{BN}(\hat{x}_i) &\sim N(\beta_i, \gamma_i^2), \end{aligned} \tag{4.6}$$

where β_i, γ_i^2 are the learned BN parameters. If the client is participating in FL or MTFL, then the model parameters W_0, b_0 are updated after downloading the global model with federated values: $\overline{W}_0$, $\overline{b}_0$. The activations of the first layer are then:

$$\begin{aligned} \overline{z}_i &\triangleq [\overline{W}_0 X + \overline{b}_0]_i, \\ \overline{z}_i &\sim N(\mathbb{E}[\overline{z}_i], \mathrm{Var}[\overline{z}_i]), \end{aligned} \tag{4.7}$$

Defining the difference in mean and variance between pre- and post-aggregation activations, $\Delta\mu_i = \mathbb{E}[\overline{z}_i] - \mathbb{E}[z_i]$ and $\Delta\sigma_i^2 = \mathrm{Var}[\overline{z}_i] - \mathrm{Var}[z_i]$, the output from a BN-patch layer as part of MTFL (which maintains μ, σ, β, γ after aggregation) is:

$$\begin{aligned} \hat{\bar{x}}_i &\sim N\left(\frac{\Delta\mu_i}{\sigma_i}, 1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right), \\ \mathrm{BN}(\hat{\bar{x}}_i) &\sim N\left(\gamma\frac{\Delta\mu_i}{\sigma_i} + \beta_i, \gamma_i^2\left(1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right)\right). \end{aligned} \tag{4.8}$$

If the BN layer is *not* a patch layer (i.e., the client is participating in FL, with federated BN values $\bar{\mu}$, $\bar{\sigma}$, $\bar{\beta}$, $\bar{\gamma}$), the output of the BN layer is:

$$\begin{aligned} \hat{\bar{x}}_i &\sim N\left(\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i}, \frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right), \\ \overline{\mathrm{BN}}(\hat{\bar{x}}_i) &\sim N\left(\bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} + \bar{\beta}_i, \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right). \end{aligned} \tag{4.9}$$

It can be posited that using BN-patch layers in MTFL constrains neuron activations to be closer to what they were before the aggregation step, compared to non-patch BN layers as part of FL, as illustrated in Figure 4.3 (right). I.e., the difference in means and variances pre- and post-aggregation using MTFL is smaller than when using FL:

$$\begin{aligned} \left|\gamma\frac{\Delta\mu_i}{\sigma_i}\right| &< \left|\beta_i - \bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} - \bar{\beta}_i\right|, \\ \left|\gamma_i^2\frac{\Delta\sigma_i^2}{\sigma_i^2}\right| &< \left|\gamma_i^2 - \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right|. \end{aligned} \tag{4.10}$$

Assuming the above inequality holds, it is easy to see how the values propagated through the network after the first layer are closer to the pre-aggregation values when using BN-patches as opposed to federated BN layers. If BN-patches are added throughout the network, the intermediate DNN values will be regularly 'constrained' to be closer to the pre-aggregation values, resulting ultimately in network outputs closer to the pre-aggregation outputs.

Inspecting (4.8), if $\Delta\mu_i$ and $\Delta\sigma_i^2$ for neuron i are large, then the output distribution of the neuron after the BN-patch layer ($\mathrm{BN}(\hat{\bar{x}})_i$) over the test-set will be quite different than $\mathrm{BN}(\hat{x})_i$. The BN-patch layer will therefore provide little benefit over a federated BN layer, as the left hand sides of the inequalities in (4.10) are unlikely to be much smaller than the right hand sides. Large differences in pre- and post-aggregated model parameters are seen during the early stages of training, when gradients are large and client models diverge more during local training. This

therefore implies that MTFL has less benefit during the early stages of training, and its benefit increases during training as gradient magnitudes decrease (as shown in Figure 4.4).

4.1.3 Federated Optimisation within MTFL

As shown in Algorithm 5, MTFL applies private patch layers for each client, and trains them alongside the federated (non-private) layers during LocalUpdate. At the end of each round, the server aggregates the uploaded federated layers from clients (and any distributed optimiser values used), producing a new global model using the GlobalModelUpdate function. If distributed adaptive-optimisation is used, then the GlobalOptimUpdate function will also be called. Table 4.1 details different FL training algorithms as characterised by their implementations of these functions.

In FedAvg, LocalUpdate is simply minibatch-SGD, and GlobalModelUpdate produces the new global model as a weighted (by number of local samples) average of uploaded client models. FedAvg uses no adaptive optimisation, so the variable V in Algorithm 5 is empty, and GlobalOptimUpdate performs no function. For FedAdam [9, 25], clients also perform SGD during LocalUpdate. However, during GlobalModelUpdate, the server takes the difference (Δ_r) between the previous global model and the average uploaded client model. The server treats Δ_r as a 'psuedogradient', and uses a set of 1$^{\text{st}}$ and 2$^{\text{nd}}$ moment values stored on the server to update the global model using an Adam-like update step. Clients do not use distributed adaptive optimisation in FedAdam, so V is also a tuple of empty values and GlobalOptimUpdate performs no function.

This chapter proposes using adaptive optimisation (namely, Adam) as the distributed optimisation strategy. This strategy is denoted as FedAvg-Adam. In FedAvg-Adam, clients share a global set of Adam 1$^{\text{st}}$ and 2$^{\text{nd}}$ moments, stored in the V variable in Algorithm 5. Clients store private optimiser values in their patch layers (W_k), as convergence performance is better when keeping private optimiser values for patches. During LocalUpdate clients perform Adam SGD, and the federated model layers and Adam values are uploaded by clients at the end of the round. To produce a new global model, the server averages the client models in GlobalModelUpdate and averages the Adam moments in GlobalOptimUpdate. FedAvg-Adam therefore has a 3× communication cost per round compared to FedAvg or FedAdam. However, in many FL scenarios, the major concern is reducing the number of communication rounds required for the model to converge. Section 4.2.2 shows that FedAvg-Adam considerably improves the convergence speed of FL and MTFL.

Table 4.1: LocalUpdate, GlobalModelUpdate and GlobalOptimUpdate used by the FedAvg [24], FedAdam [9] [25] and FedAvg-Adam FL training strategies. All of these strategies can be used within MTFL.

Optimisation Strategy	LocalUpdate	GlobalModel Update	GlobalOptim Update
FedAvg	SGD	Average	-
FedAdam	SGD	Adam	-
FedAvg-Adam	Adam	Average	Average

For the rest of this chapter, iterative FL schemes that do not keep any private model patches as are referred to as 'FL', with the optimisation strategy in brackets, e.g.

FL(FedAvg). If clients keep private model patches, the scheme is referred to as 'MTFL', again with the optimisation strategy in brackets, e.g. MTFL(FedAvg).

4.2 Experiments

This section first gives details of the datasets, models and data partitioning schemes used for all the experiments. It then presents extensive experiments analysing the impact that MTFL has on the number of rounds taken to reach a target UA. These experiments also examine which BN values, when kept private, give the best performance, and compare FL and MTFL with different optimisation strategies. After that, the reason behind why different private BN values have different impacts on training is examined, and MTFL is compared against other state-of-the-art personalised FL algorithms. Finally, the runtime of MTFL(FedAvg-Adam) is evaluated on an MEC-like testbed.

4.2.1 Datasets and Models

Experiments are conducted using two image-classification datasets: MNIST [90] and CIFAR10 [91], and two DNN architectures. The 2NN and CNN models used are the same as those described in Section 3.3.1.

Different numbers of clients W, client participation rates C and optimisation strategies, are tested for non-IID clients. To produce non-IID client data, the popular approach from [24] is used: order the training and testing data by label, split each into $2W$ shards, and assign each client two shards at random. Using the same assignment indexes for the testing data means that the classes in each client's training set are the same as those in their test set. This splitting produces a strongly non-IID distribution across clients. All results are an average over 5 trials with different random seeds.

4.2.2 Patch Layers in FL

Setup - First, the number of rounds needed to reach a target average UA (97% for MNIST, 65% for CIFAR10) for MTFL and FL are measured. In FL, no model parameters are kept private (i.e. there are no patches), whereas in MTFL, some model parameters are kept private. For the MTFL columns in Tables 2 and 3, the BN-layer statistics (μ, σ) and/or trainable parameters (γ, β) private as part of the patch layers.

For these results, the number of local epochs used was $E = 1$ and the learning rate for every scenario was tuned such that the target was reached in the fewest rounds. For FedAvg and FedAvg-Adam, only one hyperparameter was tuned for each scenario, but FedAdam requires tuning both client and server learning rates. Entries with 'X' in Tables 2 and 3 denote cases that could not reach the target within 500 communication rounds.

In Table 4.3, the robustness of MTFL to clients with noisy training data is demonstrated. Here, 20% of the clients at random had 0-mean Gaussian noise added to their training data. The average UA taken for Table 3 was for the non-noisy clients only, to test how MTFL helps to mitigate the effect of noisy clients on non-noisy clients. Gaussian noise with standard deviation 3 for MNIST and 0.2 for CIFAR10 was applied (MNIST is a much simpler image classification task than

CIFAR10, so required more noise to significantly hinder training).

Table 4.2: Communication rounds required to reach target average user accuracies for different tasks using FL and MTFL (with private statistics μ, σ and/or trained parameters γ, β), for different numbers of total clients W, client participation rates C, and optimisation strategies. Cases unable to reach the target UA within 500 rounds are denoted by X. Best results for each scenario (combination of W and C) given in bold.

	MNIST - 2NN															
	FL				**MTFL**											
	Private values = None				$\mu, \sigma, \gamma, \beta$				μ, σ				γ, β			
Optimisation	W = 200		400		200		400		200		400		200		400	
Strategy	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
FedAvg	99	102	107	110	85	58	101	68	X	X	X	X	29	21	34	26
FedAdam	85	69	88	65	56	37	75	77	109	90	194	262	31	25	31	27
FedAvg-Adam	44	49	40	50	17	41	19	32	131	151	170	198	**9**	**9**	**10**	**9**
	CIFAR10 - CNN															
FedAvg	139	138	171	164	49	33	55	36	231	280	258	266	37	24	45	30
FedAdam	105	90	83	80	21	14	22	16	67	45	48	38	24	14	25	16
FedAvg-Adam	57	43	36	31	11	9	14	**8**	82	79	62	63	**10**	**7**	**11**	**8**

Results - Table 4.2 shows that for MTFL, when all BN-layer values $(\mu, \sigma, \gamma, \beta)$ are kept private, the number of rounds to reach a target average UA is substantially lower in almost all scenarios when compared to FL. For example for the CIFAR10 $W = 400$, $C = 1.0$ scenario, FL(FedAvg) took 164 rounds to reach the target average UA, whereas MTFL(FedAvg) with private $(\mu, \sigma, \gamma, \beta)$ took only 36 rounds. However, Tables 2 and 3 show that when keeping only the tracked statistics of BN patches private, UA is actually harmed. Conversely, MTFL with only private trainable parameters took even fewer rounds than MTFL will all-private $(\mu, \sigma, \gamma, \beta)$. For the same scenario, MTFL(FedAvg) with private (μ, σ) took 266 rounds, whereas MTFL(FedAvg) with private (γ, β) took just 30 rounds. The reason behind these differences is investigated in Section 4.2.3.

MTFL naturally increases the variance of UAs during training, as non-identical client models to the variance of UAs. However in these experiments, the difference between the variance difference for FL and MTFL is small: usually less than 1%.

Table 4.3 shows that MTFL also helped to mitigate the impact of noisy clients on non-noisy clients. With FL, noisy clients prevented the average non-noisy UA from reaching the target in many scenarios. However, in most cases, MTFL allowed the non-noisy clients to reach the target average UA in a similar number of rounds than the corresponding non-noisy scenarios in Table 4.2. For example, for the CIFAR10 $W = 400, C = 1.0$ scenario, FL(FedAvg) took 250 rounds to reach the target, however MTFL(FedAvg) with private (γ, β) parameters, took just 28 rounds.

As Table 4.3 displays the rounds required for the non-noisy clients to reach the target average UA, the improvements shown when using MTFL may be due to the non-noisy clients being more 'decoupled' from the noisy ones. As they do not share all model parameters, the harmful effect of receiving a global model that has been harmed by the participation of noisy clients has been reduced, allowing them to reach higher accuracies, faster.

In most scenarios, the FedAvg-Adam optimisation strategy reached the target average UA in the fewest rounds, regardless of whether FL or MTFL is used. Taking the same CIFAR10 scenario in Table 2, FL(FedAvg) took 164 rounds, FL(FedAdam) 80 rounds, and FL(FedAvg-Adam) only 31 rounds to reach the target. Similarly, MTFL(FedAvg) took 36 rounds, MTFL(FedAdam) 16 rounds and MTFL(FedAvg-Adam) just 8 rounds with private $(\mu, \sigma, \gamma, \beta)$.

Table 4.3: Communication rounds required to reach target average user accuracies (of non-noisy clients) for different tasks using FL and MTFL (with private statistics μ, σ and/or trained parameters γ, β), when 20% of clients have noisy training data, for different numbers of total clients W, client participation rates C, and optimisation strategies. Cases unable to reach the target UA within 500 rounds are denoted by X. Best results for each scenario (combination of W and C) given in bold.

MNIST - 2NN																
	FL				**MTFL**											
	Private values = None				$\mu, \sigma, \gamma, \beta$				μ, σ				γ, β			
Optimisation	W = 200		400		200		400		200		400		200		400	
Strategy	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
FedAvg	276	X	290	X	115	76	144	144	85	58	102	68	50	36	65	48
FedAdam	X	X	X	X	76	47	110	89	56	37	75	77	43	33	46	53
FedAvg-Adam	133	260	191	X	20	16	24	27	17	41	19	32	**12**	**8**	**15**	**40**
CIFAR10 - CNN																
FedAvg	148	208	202	250	47	32	52	35	239	186	260	88	36	24	43	28
FedAdam	159	91	92	93	21	14	21	15	74	49	51	42	34	16	21	14
FedAvg-Adam	193	X	X	X	14	10	16	**9**	103	111	67	74	**12**	**8**	**13**	**9**

4.2.3 Training and Testing Results Using MTFL

Setup - To investigate why MTFL with BN patches using private (μ, σ) and/or private (γ, β) give such different results (as shown in Tables 4.2 and 4.3), we plotted training and testing UA during one scenario from Table 4.2: namely MNIST with $W = 200$, $C = 1.0$ for FL(FedAvg) and MTFL(FedAvg). The algorithms were run for 600 communication rounds, where clients performed 10 steps of local training each round, and the average training and testing UA is calculated for every local step. The graphs therefore present $600 \times 10 = 6000$ total steps. Measuring in this allows the train/test accuracy relationship, the impact that averaging has during training, and the effect on training and testing with different private BN values to be presented together.

Results - Figure 4.4 shows the (smoothed) training and testing UAs of the different combinations of BN layer statistics/parameters for the MNIST problem. Note that because the lines are smoothed for presentation, the steps where the curves reach the target accuracies may not correspond to the values in Table 4.2. In Figure 4.4 (a), the training curves for FL(FedAvg) and MTFL(FedAvg) with private (μ, σ) are the same: this is because the BN statistics are only used at test-time and do not influence training. The test accuracy for private (μ, σ) is lower than FedAvg, mirroring the results in Tables 4.2 and 4.3. The lower test accuracy may be due to mismatch in BN values: γ and β have been averaged, so output a different distribution than

these private statistics have been tracking, thus harming the ability of the model. This seems to be supported by Figure 4.4 (c). When keeping both private (μ, σ) and (γ, β) there is no substantial performance drop when compared to Figure 4.4 (b), when only (γ, β) are kept private.

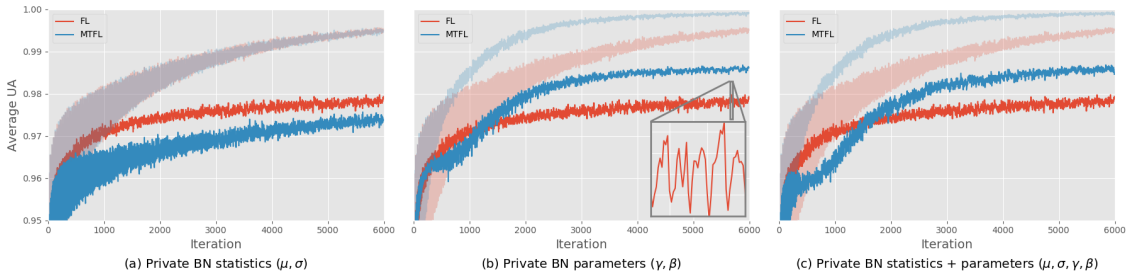


Figure 4.4: Average training (faint) and testing (solid) User Accuracy (UA) curves for every step of local SGD on the MNIST, $W = 200$, $C = 1.0$ scenario, using FL(FedAvg) (red), and MTFL(FedAvg) (blue). Each plot compares keeping different values within the BN layers of MTFL private: either statistics (μ, σ) and/or trainable parameters (γ, β), to FL. All curves have been smoothed with an averaging kernel for presentation, except the inset of plot (b), which shows the cyclic drops in accuracy due to model averaging characteristic of FL.

The results also show that keeping private (μ, σ) significantly increases the rate at which the training accuracy can improve - see faint lines in Figures 4.4 (b) and (c). Previous authors [24] have commented that FedAvg can work as a kind of regularisation for client models. When clients have small local datasets, their training error would quickly reach near-0 as it is easy for independently-trained models to overfit. However, they would have poor generalisation performance (which is the motivation behind FL). Keeping some model parameters private (here μ and σ from the BN layers) seems to strike a balance between fast convergence (which would be achieved by a fully-private model) and regularisation due to FL (which is achieved by averaging client model parameters).

4.2.4 Personalised FL Comparison

Setup - The personalisation performance of MTFL(FedAvg) is now compared with two other state-of-the-art Personalised FL algorithms: Per-FedAvg [60] and pFedMe [63], and FL(FedAvg) (where no model layers are private) [24]. The hyperparameters of each algorithm were tuned to achieve the maximum average UA within 200 communication rounds. MTFL(FedAvg) with private (γ, β) is presented rather than MTFL(FedAvg-Adam), as the personalisation algorithms should be compared, not the benefit of local adaptive optimisation. The amount of local computation is fixed to be roughly constant across the algorithms: $E = 1$ epoch of local training is performed for MTFL(FedAvg) and FL(FedAvg). For MNIST, using a batch size of 20, this is equivalent to 15 and 8 steps of local SGD for $W = 200$ and $W = 400$, respectively. For CIFAR10, this is equivalent to 13 and 7 steps of local SGD for $W = 200$ and $W = 400$, respectively. Per-FedAvg uses the value K for local iterations, so this is fixed to the same number of steps for FL and MTFL. pFedMe has two inner-loops; the number of outer-loops R is set to the same value as K from Per-FedAvg, and the number of inner-loops is fixed to 1 for all scenarios. This setup results in the same number of local steps performed for each algorithm, however the cost per local step of Per-FedAvg and pFedMe is considerably higher

than FL(FedAvg) and MTFL(FedAvg) (due to different optimisation objectives). Note also that MTFL(FedAvg) and FL(FedAvg) have only one hyperparameter, η to tune, whereas Per-FedAvg and pFedMe both have two. This makes the hyperparameter search for Per-FedAvg and pFedMe considerably more costly.

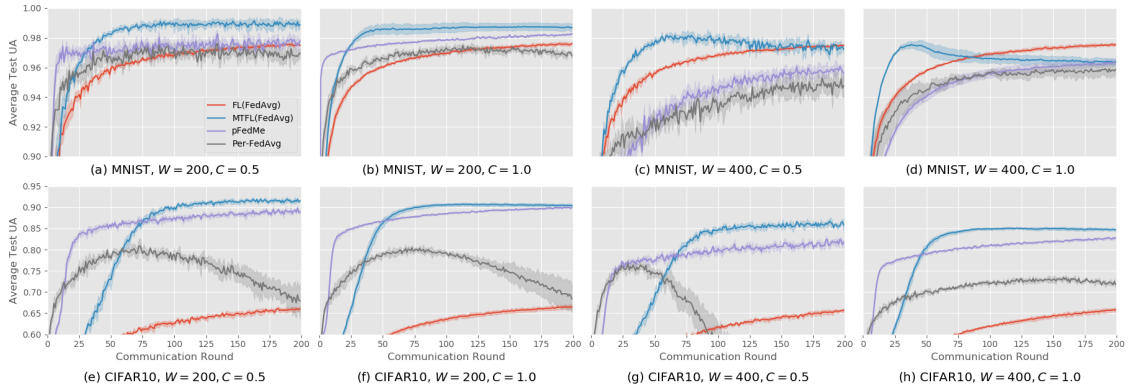


Figure 4.5: Per-round testing User Accuracy (UA) of four FL algorithms: FL(FedAvg) [24], MTFL(FedAvg) (using private γ, β), pFedMe [63] and Per-FedAvg [60]. Experiments are conducted on MNIST and CIFAR10, with data divided in a non-IID fashion between $W = 200$ or 400 clients, and $C = 0.5$ or 1.0 fraction of users participating per round. Shaded regions show 95% confidence intervals per round over 5 trials with different random seeds.

Results - The plots in Figure 4.5 show that MTFL(FedAvg) was able to achieve a higher UA compared to the other schemes in all tested scenarios. Per-FedAvg and pFedMe were able to reach a higher UA than FL(FedAvg) in the $W = 200$ cases for MNIST, but were actually slower in the $W = 400$ cases. All the personalised-FL schemes were able to achieve good UA faster than FL(FedAvg) for the CIFAR10 experiments, however. This is likely due to the CIFAR10 task being a much harder one than MNIST. It is interesting to note that Per-FedAvg appeared to overfit quickly on this task. Also worthy of note is the fact that MTFL(FedAvg) was able to beat Per-FedAvg and pFedMe whilst also having one less hyperparameter to tune, and being computationally cheaper. MTFL provides the extra benefit to privacy of keeping some model parameters private (pFedMe and Per-FedAvg both upload entire models).

4.2.5 Testbed Results

Setup - To test MTFL in a more realistic MEC environment, a testbed was set up consisting of 10 clients: 5 Raspberry Pi (RPi) 2B's and 5 RPi 3B's, connected over Wi-Fi to a server, in order to emulate a low-powered, heterogeneous set of clients. The RPi's used Tensorflow to perform local training. The server did not perform any model testing, only receiving, averaging and distributing models. The average time over 10 rounds was taken, along with the percentage of time spent per round in downloading models from the server, local training, uploading models and work performed on the server.

Results - Table 4.4 shows the average time taken per round for FL(FedAvg), MTFL(FedAvg-Adam), and Independent learning, when one local epoch of training is performed. Each round is also split by time spent for each task within the round.

As would be expected, Independent learning took the least time per round as clients did not have to download or upload any models. FL(FedAvg) took longer per round due to uploading and downloading, and MTFL(FedAvg-Adam) took the longest per round due to the increased number of weights that FedAvg-Adam communicates over FedAvg, indicated by the higher percentage of round time spent downloading and uploading models. However, the increase in communication time is likely to be outweighed in most cases by the far fewer rounds required to reach a target average UA (see Tables 2-3).

The majority of the round times were spent in local training rather than in communication for FL or MTFL. This is due to the low computing power of the RPi's and the high computational cost of training DNN models. In real-world FL scenarios, the round times are influenced by the compute abilities of client devices, the computational cost of the models used, and the communication conditions.

Table 4.4: Average time per round of different learning schemes on the MNIST and CIFAR10 datasets, and percentage of time spent downloading the model (*Down*), training the model (*Client*), uploading the model (*Up*), and model aggregation and distribution on the server (*Server*) took.

Learning Scheme	Round Time (s)	Percentage of Round Time (%) *Down*	*Client*	*Up*	*Server*
MNIST - 2NN					
FL(FedAvg)	30	5	88	6	1
MTFL(FedAvg-Adam)	38	11	76	12	1
Independent	29	0	100	0	0
CIFAR10 - CNN					
FL(FedAvg)	108	5	86	5	4
MTFL(FedAvg-Adam)	136	11	74	12	3
Independent	100	0	100	0	0

4.3 Chapter Summary

A Multi-Task Federated Learning (MTFL) algorithm was proposed that builds on iterative FL algorithms by introducing private patch layers into the global model. Private layers allow users to have personalised models and significantly improves average User model Accuracy (UA). The use of BN layers as patches in MTFL was analysed, providing insight into the source of their benefit. MTFL is a general framework that requires a specific FL optimisation strategy, and the FedAvg-Adam optimisation scheme (which uses Adam on clients) was also proposed. Experiments using MNIST and CIFAR10 showed that MTFL with FedAvg significantly reduces the number of rounds to reach a target average UA compared to FL, by up to 5×. Further experiments show that MTFL with FedAvg-Adam reduces this number even further, by up to 3×. These experiments also indicate that using private BN trainable parameters (γ, β) instead of statistics (μ, σ) in model patches gives better convergence speed. Comparison to other state-of-the-art personalised FL algorithms show that MTFL is able to achieve the highest average UA given limited communication rounds. Lastly, experiments using a MEC-like testbed showed that the communication overhead of MTFL with FedAvg-Adam is outweighed by its significant benefits over FL with FedAvg in terms of UA and convergence speed.

Chapter 5

Faster Federated Learning with Decaying Number of Local SGD Steps

As shown in Chapter 2, FedAvg can improve its communication-efficiency by performing more steps of SGD on clients between rounds of model averaging [24]. Improving the communication-efficiency of FL is highly important due to the low-bandwidth connections of devices at the network edge, leading to very long training times. However real-world client data is highly heterogeneous, which has been extensively shown to slow model convergence. Furthermore, when $K > 1$ steps of SGD are performed on clients per round the final performance of the FL model is harmed due to client-drift [26], and the influence of local compute time on the total training time becomes more significant [97]. This chapter proposes decaying K as training progresses, which can jointly improve the final performance of the global model whilst reducing the wall-clock time to reach a given model error and the total computational cost of training, compared to using a fixed value of K throughout. FedAvg's convergence on strongly-convex objectives is analysed with a decaying value of K, providing novel insights into the convergence properties of this scheme, and motivating three practical decay schedules for K based respectively on the communication round number, the relative model error, and the validation performance. Thorough experiments are then performed on four benchmark FL datasets (FEMNIST, CIFAR100, Sentiment140, Shakespeare) using realistic edge bandwidth and computation-time values, which show the benefits of the K-decay schedules in terms of convergence time, computational cost, and generalisation performance.

5.1 FedAvg with Decaying Local Steps

This section formally describes the FL optimisation problem, theoretically analyses the convergence of FedAvg with a decaying number of local SGD steps, and present theoretically-motivated schedules based upon the analysis.

5.1.1 Problem Setup

In FL there are a large number of clients that each possess a small number of local samples. The objective is to train model $\boldsymbol{x}$ to minimise the expected loss over all

samples and over all clients, namely:

$$F(\boldsymbol{x}) = \sum_{c=1}^{C} p_c f_c(\boldsymbol{x}) = \sum_{c=1}^{C} p_c \left[\sum_{n=1}^{n_c} f(\boldsymbol{x}, \xi_{c,n}) \right], \tag{5.1}$$

where C is the total number of FL clients, p_c is the fraction of all samples owned by client c (such that $\sum_{c=1}^{C} p_c = 1$), f is the loss function used on clients, and $\{\xi_{c,1}, \cdots, \xi_{c,n_c}\}$ represent the training samples owned by client c.

To minimise $F(\boldsymbol{x})$ in a communication-efficient manner as discussed in Section 2.3, FedAvg (presented in Algorithm 1) performs multiple steps of SGD on each client between model averaging, as discussed in Section 2.3. The updates to clients models within the FedAvg process can be viewed from the perspective of communication rounds (as shown in most FL works and presented in Algorithm 1), but can also be reformulated in terms of a continuous sequence of SGD steps on each client, with updates periodically being replaced by averaging. Suppose the client model iterations are reindexed from $\boldsymbol{x}_{r,k}^c$ to $\boldsymbol{x}_t^c$ where t is the *global iteration*, $t \in \{1, \cdots, T\}$. Note that $\sum_r \{K_r\} = T$. For a given FL client i and local SGD step t the update to the local model $\boldsymbol{x}_t^i$ can be given as:

$$\begin{aligned} \boldsymbol{y}_{t+1}^i &= \boldsymbol{x}_t^i - \eta_t \nabla f(\boldsymbol{x}_t^i, \xi_t^i), \\ \boldsymbol{x}_{t+1}^i &= \begin{cases} \sum_{c \in \mathcal{C}_t} p_c \boldsymbol{y}_{t+1}^c & \text{if } t \in \mathcal{I}, \\ \boldsymbol{y}_{t+1}^i & \text{otherwise}, \end{cases} \end{aligned} \tag{5.2}$$

where $\mathcal{I}$ is the set of indexes denoting the iterations at which model communication occurs (which will be equal to the cumulative sum of $\{K_r\}$). This formulation states that clients not participating in the current round compute and then discard some local updates, which is not true in reality but makes analysis more amenable and is theoretically equivalent to FedAvg as presented in Algorithm 1. The average client model at any given iteration t using (2) is defined as: $\bar{\boldsymbol{x}}_t = \sum_{c=1}^{C} p_c \boldsymbol{x}_c^t$.

5.1.2 Runtime Model of FedAvg

Inspecting Algorithm 1 shows that the nominal wall-clock time for each client c to complete a communication round r is:

$$W_r^c = \frac{|\boldsymbol{x}|}{D^c} + K_r \beta^c + \frac{|\boldsymbol{x}|}{U^c}, \tag{5.3}$$

where $|\boldsymbol{x}|$ is the size of the FL model (in megabits), U^c and D^c are the upload and download bandwidth of client c in megabits per second, and β^c is the per-minibatch computation time of client c. The nominal time to complete a round for client c is therefore the sum of the download, local compute, and upload times. Furthermore, as FL clients are usually connected wirelessly at the network edge and geographically dispersed, the runtime model assumes that U^c and D^c are independent of the total number of participating clients. For wireless connections, typically $D^c >> U^c$ [18].

For a single round, the server must wait for the slowest client (straggler) to send its update. Therefore the time taken to complete a single round W_r is:

$$W_r = \max_{c \in \mathcal{C}_r} \{W_r^c\}. \tag{5.4}$$

To simplify the FedAvg runtime model, the runtime model assumes that all clients have the same upload bandwidth, download bandwidth, and per-minibatch compute

time. That is, $U^c = U$, $D^c = C$, and $\beta^c = \beta$, $\forall\ c$. Using these simplifications, the total runtime W for R communication rounds of FedAvg are:

$$W = \sum_{r=1}^{R} W_r = R\left(\frac{|\boldsymbol{x}|}{D} + \frac{|\boldsymbol{x}|}{U}\right) + \beta \sum_{r=1}^{R} K_r. \tag{5.5}$$

Previous works consider a fixed number of local steps during training: $K_r = K$, $\forall\ r$. There are extensive works showing that a larger K can lead to an increased convergence rate of the global model [24, 30]. However, large K means that fewer communication rounds can be completed in a given timeframe. Previous works have shown that due to the low computational power of FL clients, the value of β can dominate the per-round runtime [97]. Therefore by decaying K_r during training, a balance between fast convergence and higher round-completion rate can be achieved, which is the primary focus of this chapter.

5.1.3 Convergence Analysis

A convergence analysis of FedAvg using a decaying number of local steps K_r and constant learning rate η is now presented for strongly-convex objective functions. The following typical assumptions for theoretical analysis within FL are made.

Assumption 1: Client objective functions are L-smooth: $f_c(\boldsymbol{x}) \leq f_c(\boldsymbol{y}) + (\boldsymbol{x} - \boldsymbol{y})^\top \nabla f_c(\boldsymbol{y}) + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2$. As $F(\boldsymbol{x})$ is a convex combination of f_c, then it is also L-smooth.

Assumption 2: Client objective functions are μ-strongly convex: $f_c(\boldsymbol{x}) \geq f_c(\boldsymbol{y}) + (\boldsymbol{x} - \boldsymbol{y})^\top \nabla f_c(\boldsymbol{y}) + \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2$, with minima $f_c^* = \min f_c$. As $F(\boldsymbol{x})$ is a convex combination of f_c, then it is also μ-strongly convex.

Assumption 3: For uniformly sampled data points ξ_k^c on client c, the variance of stochastic gradients on c are bounded by: $\mathbb{E}\|\nabla f_c(\boldsymbol{x}; \xi_k^c) - \nabla f_c(\boldsymbol{x})\|^2 \leq \sigma_c^2$.

Assumption 4: The expected squared norm of stochastic gradients on all clients is bounded: $\mathbb{E}\|\nabla f_c(\boldsymbol{x}); \xi_k^c\|^2 \leq G^2, \forall c$.

As per [28], the extent of non-IID client data is quantified with: $\Gamma = F^* - \sum_{c=1}^{C} p_c f_c^*$, where F^* is the minimum point of $F(\boldsymbol{x})$. $\Gamma \neq 0$ when the minimiser of the global objectives is not the same as the average minimiser of client objectives. $\Gamma = 0$ if the FL data is IID over the clients.

Theorem 5.1: *Let Assumptions 1-4 hold, and define $\kappa = L/\mu$. The expected minimum gradient norm of FedAvg using a monotonically decreasing number of local SGD steps K_r and fixed stepsize $\eta \leq 1/4L$ after T total iterations is given by:*

$$\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\} \leq \frac{2\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{\eta T} + \eta\kappa L\left[\sum_{c=1}^{C} p_c^2 \sigma_c^2 + 6L\Gamma + \left(8 + \frac{4}{N}\right) G^2 \frac{\sum_{r=1}^{R} K_r^3}{\sum_{r=1}^{R} K_r}\right]. \tag{5.6}$$

Proof: See Appendix A.2.

Theorem 5.1 shows that with a fixed learning rate and decreasing K_r, FedAvg converges with $\mathcal{O}(1/T)+\mathcal{O}(\eta)$. This result reflects the classical result of centralised SGD with a fixed learning rate (albeit with different constants due to non-IID clients and $K_r > 1$). Previous works have shown (like in centralised SGD) the requirement for η to be decayed to allow FedAvg to converge arbitrarily close to the global minima [26, 28]. However this chapter is concerned with the runtime and computational savings available when decaying K_r, so does not re-prove the already-covered decaying η result.

The second term of Theorem 5.1 shows that using $K_r > 1$ harms the convergence of FedAvg in terms of total number of iterations T. This is the case for all state-of-the-art analyses save for quadratic objectives [76]. However in FL minimising the number of communication rounds/quantity of communicated data alongside the total number of iterations (both of which affect the runtime of FedAvg) is highly important. When using a decreasing η, previous analyses have shown that $K > 1$ acts to reduce the variance introduced by client stochastic gradients (the $\sum_{c=1}^{C} p_c^2\sigma_c^2$ term) [26, 28]. Dividing (6) by K_r (to achieve the convergence result in terms of total number of rounds) shows the same benefit in the above analysis. It is observed empirically that $K_r > 1$ helps to reduce the variance of client updates even with a fixed η. Similarly a large number of clients participating per round (N) helps to reduce the variance that is introduced by performing $K_r > 1$ steps. FedAvg deployments can therefore benefit more from sampling a larger number of clients per round N when the number of local steps K_r is large.

This chapter's formulation of FedAvg and its analysis assumes a constant participation rate, but in real-world FL the round participation rate varies [8]. Setting K_r to a large value makes more progress in a round, but fewer clients will be able to complete the round in a given timeframe. This poses an interesting trade-off between K_r and N, which could be a potential avenue for future research.

5.1.4 Optimal Values of K_r and η_r

When using a fixed number of local steps $K_r = K$ and communication rounds R, the total number of FedAvg iterations is $T = KR$. Substituting this into (5.5) gives the total runtime W of T iterations of FedAvg:

$$W = \frac{T}{K}\left[\frac{|\boldsymbol{x}|}{D} + \frac{|\boldsymbol{x}|}{U} + \beta K\right]. \tag{5.7}$$

Setting $K_r = K$ and substituting (5.7) into Theorem 5.1 gives the convergence of FedAvg for fixed K and η in terms of the runtime, rather than the number of iterations:

$$\begin{aligned}\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\} \leq \frac{2\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{\eta W K}\left[\frac{|\boldsymbol{x}|}{D} + \frac{|\boldsymbol{x}|}{U} + \beta K\right] \\ + \eta\kappa L\left[\sum_{c=1}^{C} p_c^2\sigma_c^2 + 6L\Gamma + \left(8 + \frac{4}{N}\right)G^2K^2\right].\end{aligned} \tag{5.8}$$

The optimal value of K to minimise (5.8) can then be derived for any given point during the runtime of FedAvg.

Theorem 5.2: *Let Assumptions 1-4 hold and define* $\kappa = L/\mu$. *For fixed* $\eta \leq 1/4L$,

the optimal number of local SGD steps K to (8) is given by:

$$K^* = \sqrt[3]{\frac{(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{8\eta^2 L\,(1 + {}^1\!/_{2N})} \frac{({}^{|\boldsymbol{x}|}\!/_D + {}^{|\boldsymbol{x}|}\!/_U)}{W}}. \tag{5.9}$$

Proof: See Appendix A.3.

Theorem 5.2 shows that the optimal value of K decreases with $\mathcal{O}({}^1\!/\sqrt[3]{W})$. Wang and Joshi [98] investigated variable communication intervals for the Periodic Averaging SGD (PASGD) algorithm in the datacentre, and found that the optimal interval decreased with $\mathcal{O}({}^1\!/\sqrt[2]{W})$. The optimal value of K decreases slower in FedAvg due to the looser bound on client divergence between averaging (scaling with K^2 rather than K). As the number of clients participating per round (N) increases, the optimal value of K increases. This is because a higher number of participating clients decreases the variance in model updates (which is especially significant considering the non-IID client data).

The relationship between the communication time and computation time is more complex. In FL, it is typically assumed that the local computation time is dominated by the communication time due to the low-bandwidth connections to the coordinating server. Considering the case where $({}^{|\boldsymbol{x}|}\!/_D + {}^{|\boldsymbol{x}|}\!/_U >> \beta K)$, then $W \approx R\,({}^{|\boldsymbol{x}|}\!/_D + {}^{|\boldsymbol{x}|}\!/_U)$. This means:

$$K^* = \sqrt[3]{\frac{\kappa F(\bar{\boldsymbol{x}}_0) - F^*}{8\eta^2 L\,(1 + {}^1\!/_{2N})} \frac{1}{R}}, \tag{5.10}$$

i.e. the optimal value of K is not dependent on the local computation time, only the total number of rounds R. This decay scheme produces a fairly aggressive decay rate, and is tested experimentally in Section 5.2 using a variety of model types (with different communication and computation times).

A similar approach can be taken to find the optimal value of η_r at each communication round. Although the focus of this chapter is on decaying K to improve the convergence speed of FL, it is compares to the effect of decaying η as well as constant η/K.

Corollary 5.2.1: *Let Assumptions 1-4 hold and define $\kappa = {}^L\!/_\mu$. Given stepsizes $\eta_r \leq {}^1\!/_{4L}$, the optimal value of η to minimise (8) is given by:*

$$\eta^* = \sqrt{\frac{2(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{LZ} \frac{({}^{|\boldsymbol{x}|}\!/_D + {}^{|\boldsymbol{x}|}\!/_U + \beta K)}{W}}, \tag{5.11}$$

where $Z = \sum_{c=1}^{C} p_c^2 \sigma_c^2 + 6L\Gamma + (8 + {}^4\!/_N) G^2 K^2$.

Proof: See Appendix A.4.

Corollary 5.2.1 shows that η^* is directly affected by the per-round time: as any of the upload, download or computation time increases, η^* increases. This is because less progress is made over time (due to longer rounds) so a larger η^* compensates by making more progress per SGD step. Similar to K^*, a larger number of clients participating per round (N) allows for a smaller η^* by reducing variance due to client-drift. Larger K also allows for smaller η as more progress is made per round.

Making the same assumption as in (5.10) ($|\boldsymbol{x}|/D + |\boldsymbol{x}|/U >> \beta K$) gives a decay schedule for η_r in terms of the number of communication rounds:

$$\eta^* = \sqrt{\frac{2(\kappa F(\boldsymbol{x}_0) - F^*)}{LZ}\frac{1}{R}}, \tag{5.12}$$

where Z is defined in (5.11). This schedule is also tested empirically in Section 5.2.

5.1.5 Schedules Based on Training Progress

In practice the values of κ, F^*, and L are difficult or impossible to evaluate due to complex nonlinear models (i.e. DNNs) and data privacy in FL. Therefore, appropriate values of K and η are chosen via grid-search or some other method (such as Bayesian Optimisation). Denote K_0 as a 'good' value of K at $W = 0$ (found via grid search), and K_r as the value of K to be used for round r. Each successive round of FedAvg can be considered as a new optimisation procedure with starting model $\bar{\boldsymbol{x}}_r$. If the further assumption that $F^* = 0$ is made, substituting these two sets of values into (5.9) and dividing gives K_r in terms of K_0:

$$K_r = \left\lceil \sqrt[3]{\frac{F(\bar{\boldsymbol{x}}_r)}{F(\bar{\boldsymbol{x}}_0)}}\, K_0 \right\rceil. \tag{5.13}$$

A similar process can be applied to find η_r in terms of η_0:

$$\eta_r = \sqrt[2]{\frac{F(\bar{\boldsymbol{x}}_r)}{F(\bar{\boldsymbol{x}}_0)}}\, \eta_0. \tag{5.14}$$

$F(\bar{\boldsymbol{x}}_r)$ is the training loss of the global model at the start of round r. Practically, an estimate of $F(\bar{\boldsymbol{x}}_r)$ can be obtained by requiring clients $c \in \mathcal{C}_r$ to send their training loss after the first step of local SGD to the server each round: $f_c(\bar{\boldsymbol{x}}_r, \xi_{c,0})$, where $\mathbb{E}\left[f_c(\bar{\boldsymbol{x}}_r, \xi_{c,0})\right] = F(\bar{\boldsymbol{x}}_r)$. This is only a single floating-point value that does not require any extra computation and negligibly increases the per-round communication costs.

Due to only a small fraction of the non-IID clients being sampled each round, the per-round variance of $\frac{1}{N}\sum_{c\in\mathcal{C}_r} f_c(\bar{\boldsymbol{x}}_r, \xi_{c,0})$ can be very high. Therefore, a simple rolling-average estimate using window size s can be used:

$$F(\bar{\boldsymbol{x}}_r) \approx \frac{1}{sN}\sum_{i=r-s}^{r}\sum_{c\in\mathcal{C}_i} f_c(\bar{\boldsymbol{x}}_i, \xi_{c,0}). \tag{5.15}$$

The experiments in Section 5.2 use a window size $s = 100$, where the experiments run for at least $R = 10,000$ communication rounds. For the first s rounds when (5.15) cannot be computed, the experiments keep $K_r = K_0$.

When using a fixed value of K, Theorem 5.1 shows that the minimum gradient norm converges with $\mathcal{O}(1/T) + \mathcal{O}(\eta K^2)$. As noted earlier, this result is analogous to the classical result of dSGD using a fixed learning rate. In the datacentre, the practical heuristic of decaying the learning rate η when the validation error plateaus is commonly used to allow the model to reach a lower validation error. A similar strategy for FedAvg can therefore be used: once the validation error plateaus decay either K or η. This heuristic is investigated alongside the decay schedules presented above in Section 5.2.

5.2 Experimental Evaluation

In this section, the results of simulations comparing the three decaying-K schemes proposed in Section 5.1 are presented, demonstrating their benefits in terms of runtime, communicated data and computational cost on four benchmark FL datasets. Table 5.1 overviews the learning tasks, and they are described in more detail in Section 5.2.1.

Table 5.1: Datasets and models used in the experimental evaluation (DNN = Deep Neural Network, CNN = Convolutional Neural Network, GRU = Gated Recurrent Network, SA = Sentiment Analysis, CV = Computer Vision, NLP = Natural Language Processing). K_0 and η_0 are the initial number of local steps and initial learning rate used.

Task	Type	Classes	Model	Model Size (Mb)	Total Clients	Clients per Round	Samples per Client	K_0	η_0
Sent140	SA	2	Linear	0.32	21876	50	15	60	3.0
FEMNIST	CV	62	DNN	6.71	3000	60	170	80	0.3
CIFAR100	CV	100	CNN	40.0	500	25	100	50	0.01
Shakespeare	NLP	79	GRU	5.21	660	10	5573	80	0.1

5.2.1 Datasets and Models

To show the broad applicability of the K-decay approach, experiments are conducted on 4 benchmark FL learning tasks from 3 Machine Learning domains (sentiment analysis, computer vision, natural language processing) using 4 different model types (simple linear, DNN, Convolutional, Recurrent).

Sentiment 140: a sentiment analysis task of Tweets from a large number of Twitter users [99]. This dataset is limited to users with ≥ 10 samples, leaving 22k total clients, with 336k training and 95k validation samples, and an average of 15 training samples per client. A normalised bag-of-words vector of size 5k was generated for each sample using the 5k most frequent tokens in the dataset. A binary linear classifier using 50 clients per round (0.2% of all clients) and a batch size of 8 was trained.

FEMNIST: an image classification task of (28×28) pixel greyscale (flattened) images of handwritten letters and numbers from 62 classes, grouped by the writer of the symbol [99]. 3k total clients were used, with a total of 501k training and 129k validation samples and an average of 170 training samples per client. 60 clients were selected per round (2% of all clients) with a batch size of 32. A DNN comprising a 200-unit ReLU Fully-Connected layer (FC), a second 200-unit ReLU FC layer, and a softmax output layer was trained.

CIFAR100: an image classification task of (32×32) pixel RGB images of objects from 100 classes. The non-IID partition first proposed in [25] was used, which splits the images into 500 clients based on the class labels. There are 50k training and 10k validation samples in the dataset, with each client possessing 100 samples. 25 clients per round are selected (5% of all clients). A Convolutional Neural Network (CNN) consisting of two (3×3) ReLU convolutional + (2×2) Max-Pooling blocks, a 512-unit ReLU FC layer, and a softmax output layer was trained with a batch size of 32. As per other FL works [25, 27] random preprocessing was applied to improve

generalisation, composed of a random horizontal flip and crop of the (28×28) pixel sub-image.

Shakespeare: a next-character prediction task using the complete plays of William Shakespeare [99]. The lines from all plays are partitioned by the speaking part in each play, and clients with ≤ 2 lines are discarded, leaving 660 total clients. Using a sequence length of 80, there are 3.7m training and 357k validation samples, with an average of 5573 training samples per client. 10 clients per round are selected (1.5% of all clients) with a batch size of 32. A Gated Recurrent Unit (GRU) was trained comprising a $79 \rightarrow 8$ embedding, two stacked GRUs of 128 units, and a softmax output layer.

5.2.2 Simulating Communication and Computation

The convergence of FedAvg for the above learning tasks was simulated using PyTorch on GPU-equipped workstations. However, real-world FL runs distributed training on low-powered edge clients (such as smartphones and IoT devices). These clients exhibit much lower computational power and lower bandwidth to the coordinating server compared to datacentre nodes.

To realistically simulate real-world FedAvg, the runtime model presented in Section 5.1.2 and Equation (5.5) was used. Each client is give a download bandwidth of $D = 20$ Mbps and an upload bandwidth of $U = 5$ Mbps. These are typical values for wireless devices connected via 4G LTE in the United Kingdom [18]. To determine the runtime of a minibatch of SGD on a typical low-powered edge device (β), 100 steps of minibatch-SGD for each learning task was run on a Raspberry Pi 3B+ with the following configuration:

- 1.4GHz 64-bit quad-core Cortex-A53 processor.
- 1GB LPDDR2 SDRAM.
- Ubuntu Server 22.04.1.
- PyTorch 1.8.2.

As shown in Table 5.2, there is a large difference in the minibatch runtimes between the tasks. This is due to the relative computational costs of the models used: the Sent140 task uses a simple linear model, whereas the Shakespeare GRU model requires a far larger number of matrix multiplications for a single forward-backward pass.

Table 5.2: Mean and standard deviation of runtimes for a minibatch of SGD for each learning task using a Raspberry Pi 3B+.

Task	Mean (s)	Std (s)
Sent140	5.2×10^{-3}	2.1×10^{-4}
FEMNIST	0.017	5.1×10^{-4}
CIFAR100	0.31	1.7×10^{-2}
Shakespeare	1.5	8.5×10^{-2}

5.2.3 K_r and η_r Decay Schedules

For each learning task, FedAvg was run for 10k communication rounds using fixed $K_r = K_0$ and $\eta_r = \eta_0$ (henceforth '$K\eta$-fixed'). The number of rounds reflects typical real-world deployments (which are on the order of thousands of rounds) [8]. K_0 and η_0 were selected via grid-search such that the validation error for each task could plateau within the 10k rounds, with the values presented in Table 5.1. dSGD (FedAvg with $K_r = 1$) was also run to show the runtime benefit of using $K > 1$ local steps.

FedAvg was then run sing the three schedules for K_r and the three schedules for η_r as discussed in Section 5.1.4 and 5.1.5. Table 5.3 shows the different decay schedules tested and the name used to denote each one by in Section 5.2.4. Jointly decaying K_r *and* η_r during training was also tested. However decaying either K_r or η_r decreases the amount of progress that is made during each training round as the global model changes less. It was found empirically that decaying both lead to training progress slowing too rapidly, so these results are not included in Section 5.2.4.

Table 5.3: Values of K_r and η_r for given communication round r as tested in the experimental evaluation.

Schedule	K_r	η_r
dSGD	1	η_0
$K\eta$-fixed	K_0	η_0
K_r-rounds (10)	$\lceil \sqrt[3]{1/r}\ K_0 \rceil$	η_0
K_r-error (13)	$\lceil \sqrt[3]{F_r/F_0}\ K_0 \rceil$	η_0
K_r-step	$K_0/10$ if converged	η_0
η_r-rounds (12)	K_0	$\sqrt[2]{1/r}\ \eta_0$
η_r-error (14)	K_0	$\sqrt[2]{F_r/F_0}\ \eta_0$
η_r-step	K_0	$\eta_0/10$ if converged

5.2.4 Results

Figure 5.1 shows the minimum cumulative training error achieved by FedAvg for the different K_r and η_r schedules (as shown in Table 5.3). Confidence intervals for Figure 5.1 were omitted for clarity due to the large number of curves. For all tasks other than Shakespeare, FedAvg with $K\eta$-fixed (solid grey curve) increases the convergence rate compared to dSGD (dashed grey curve). For Shakespeare (Figure 5.1 (d)), $K\eta$-fixed improved the initial convergence rate but was overtaken by dSGD at approximately 2500 minutes. This is likely because of the very high computation time for Shakespeare (see Table 5.2) relative to the other datasets (due to the very high computational cost of the GRU model).

For Sentiment 140 (Figure 5.1 (a)) and Shakespeare (Figure 5.1 (d)), decaying either K_r or η_r during training lead to smaller improvements in the training error that was achieved. However, for FEMNIST and CIFAR100, the K_r-rounds scheme lead to lower training error compared to $K\eta$-fixed. For CIFAR100, an improvement was also seen with η_r-rounds. Both FEMNIST and CIFAR100 are image classification tasks, so it be may the case that decaying K_r or η_r during training is beneficial

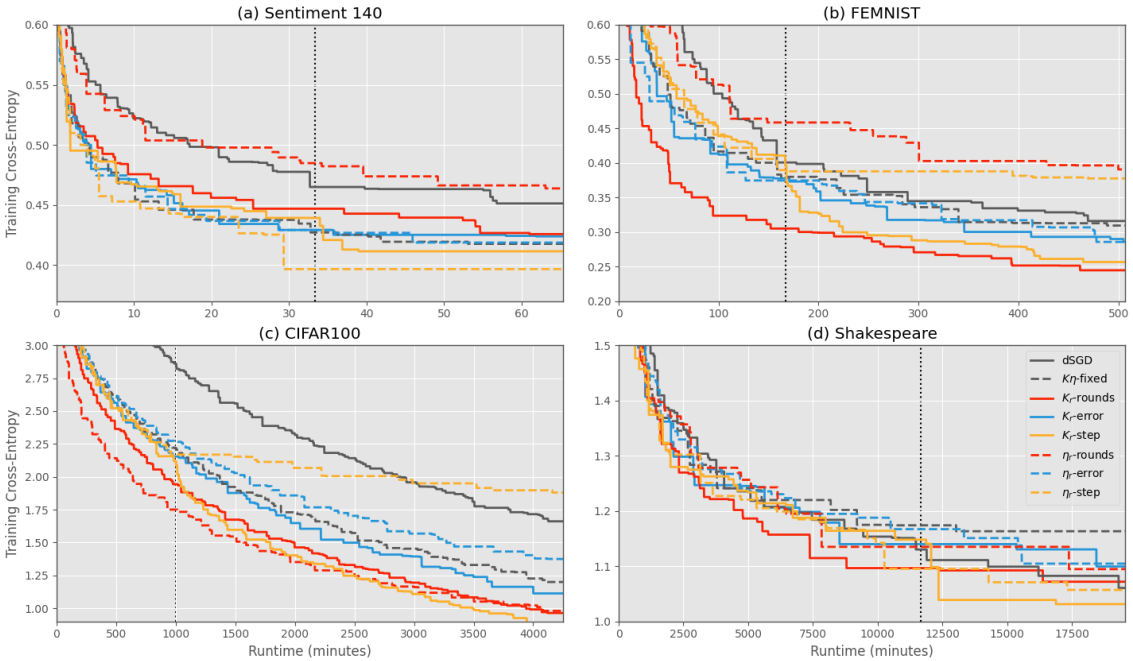


Figure 5.1: Cumulative lowest training cross-entropy error over time of FedAvg using different schedules for K_r and η_r. Curves show mean over 5 random trials. Vertical line shows the communication round where the validation error plateaus.

for computer vision, which could be investigated further in future works.

Figure 5.2 shows the impact on validation accuracy for the tested decay schedules. The $K\eta$-fixed schedule shows faster initial convergence for all tasks, but it is overtaken by dSGD in the later stages of training. For FEMNIST, CIFAR100 and Shakespeare, the aggressive K_r-rounds and K_r-step schemes improved the convergence rate compared to dSGD, with very significant improvement for CIFAR100. A marked increase in convergence rate can be seen in Figure 5.1 (c) at 1000 minutes when K_r-step is decayed.

In all tasks, all K-decay schemes were able to match or improve the validation accuracy that $K\eta$-fixed achieved whilst performing (often substantially) fewer total steps of SGD within a given runtime. Table 5.4 shows the total SGD steps performed by the K-decay schemes relative to the total steps performed by $K\eta$-fixed over the 10k communication rounds (all the η-decay schemes perform the same amount of computation as $K\eta$-fixed). The fact that K-decay schemes can outperform $K\eta$-fixed with lower total computation indicates that much of the extra computation performed by FedAvg is wasted when considering validation performance. CIFAR100 using K_r-rounds for example achieved over 18% higher validation accuracy compared to $K\eta$-fixed whilst performing less than 10% of the total steps of SGD. Similarly, K_r-step achieved the same validation accuracy as $K\eta$-fixed whilst performing only 68% of the total SGD steps.

5.3 Chapter Summary

The FedAvg improves the convergence rate of the global FL model by performing several steps of SGD (K) locally during each training round. In this chapter, FedAvg was theoretically analysed to examine the runtime benefit of decreasing (K) during training. A runtime model of FedAvg was set up and used to determine

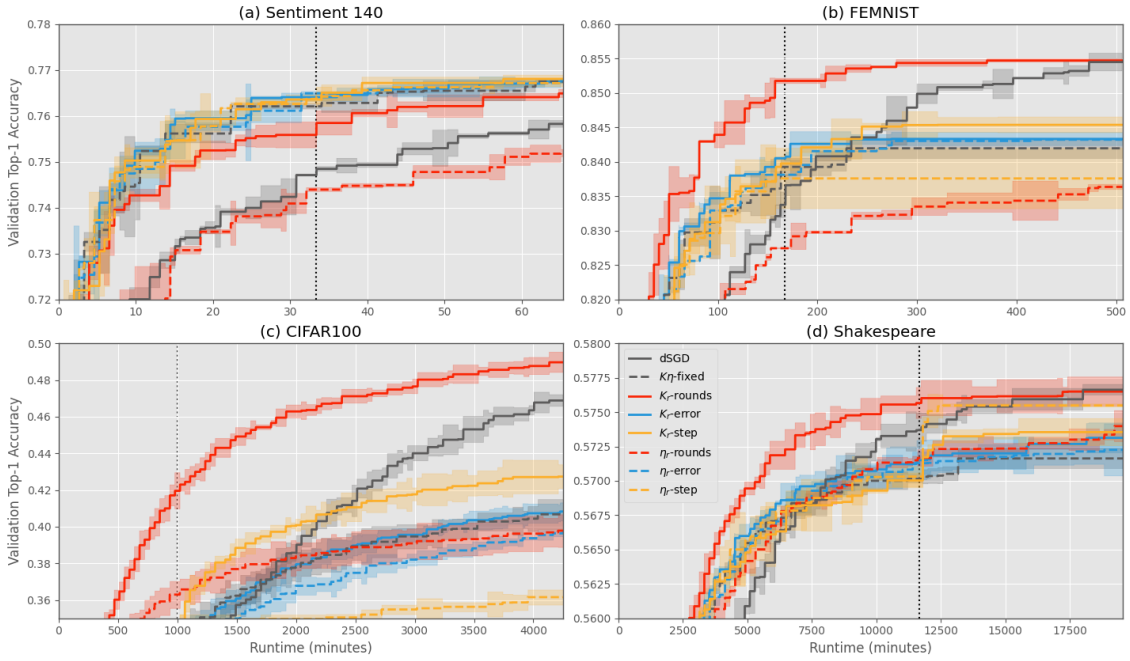


Figure 5.2: Cumulative highest validation top-1 accuracy over time of FedAvg using different schedules for K_r and η_r. Curves show mean and shaded regions show 95% confidence intervals of the mean over 5 random trials.

Table 5.4: Total SGD steps performed during training for each K-decay schedule relative to $K\eta$-fixed for different learning tasks.

Task	Schedule	Relative SGD Steps			
		Sentiment 140	FEMNIST	CIFAR100	Shakespeare
Sentiment 140	K_r-rounds	0.21	0.11	0.090	0.74
	K_r-error	0.99	0.80	0.57	0.99
	K_r-step	0.68	0.44	0.40	0.96

the optimal value of K (and learning rate η) at any point during training under different assumptions, leading to three practical schedules for decaying K as training progresses. Simulated experiments using realistic values for communication-time and computation-time on 4 benchmark FL datasets from 3 learning domains showed that decaying K during training can lead to improved training error and validation accuracy within a given timeframe, in some cases whilst performing over $10\times$ less total computation compared to fixed K.

Chapter 6

Accelerating Federated Learning with a Global Biased Optimiser

Adaptive optimisation can significantly improve the convergence rate of the global model, reducing FL's long training times. Chapters 3 and 4 both propose novel ways of using adaptive optimisation in FL. However, these approaches allow optimiser values to change within the client update loop, which could potentially contribute to client-drift [27] and increases the per-round upload cost compared to FedAvg (as unique optimiser values must be sent to the coordinating server). In this chapter, a novel and generalised approach for incorporating adaptive optimisation in FL is proposed with the Federated Global Biased Optimiser (FedGBO) algorithm. FedGBO accelerates FL by employing a set of global biased optimiser values during training, reducing client-drift and client model variance due non-IID data whilst benefiting from adaptive optimisation. The updates to the global model in FedGBO can be reformulated as centralised training updates using biased gradients and optimiser updates, and this framework is used to prove FedGBO's convergence on nonconvex objectives when using heavy-ball momentum-SGD (SGDm) [100]. Extensive experiments using 4 FL benchmark datasets (CIFAR100, Sent140, FEMNIST, Shakespeare) and 3 popular optimisers (SGDm, RMSProp [101], Adam [87]) are conducted to compare FedGBO against six state-of-the-art FL algorithms. The results demonstrate that FedGBO displays superior or competitive performance across the datasets whilst having low data-upload and computational costs, and provide practical insights into the trade-offs associated with different adaptive-FL algorithms and optimisers.

6.1 The FedGBO Algorithm

6.1.1 Algorithm Design

The Federated Global Biased Optimiser (FedGBO) algorithm operates in rounds similar to FedAvg. In each communication round of FedGBO (Algorithm 6), a random subset of clients are selected to participate (line 3). This selection forces partial client participation in the experimental setting, however in a real-world deployment, clients devices will participate if they are available for training (e.g., for smartphone clients this may be when they are charging and connected to Wi-Fi) [8]. FedGBO is a generic FL algorithm compatible with any adaptive optimiser that can be represented as an update ($\mathcal{U}$), tracking ($\mathcal{T}$), and inverse ($\mathcal{I}$) step. Table 6.1 presents $\mathcal{U}, \mathcal{T}, \mathcal{I}$ for three popular optimisers: RMSProp [101], SGDm [100], and Adam [87].

Algorithm 6: Federated Global Biased Optimiser (FedGBO)

1 **input:** initial global model $\boldsymbol{x}_0$, optimiser $\boldsymbol{s}_0$
2 **for** round $r = 1$ **to** R **do**
3 select round clients $\mathcal{C}_r$
4 **for** client $c \in \mathcal{C}_r$ in parallel **do**
5 download global model $\boldsymbol{x}_t$, and optimiser $\boldsymbol{s}_t$
6 initialise local model, $\boldsymbol{y}_0^i \leftarrow \boldsymbol{x}_t$
7 **for** local SGD step $k = 1$ **to** K **do**
8 compute minibatch gradient $\boldsymbol{g}_t^i$
9 $\boldsymbol{y}_k^i \leftarrow \boldsymbol{y}_{k-1}^i - \eta \mathcal{U}(\boldsymbol{g}_t^i, \boldsymbol{s}_t)$
10 **end**
11 upload local model $\boldsymbol{y}_K^i$ to server
12 **end**
13 update global model $\boldsymbol{x}_{t+1} \leftarrow \frac{1}{|S_t|} \sum_{i \in S_t} \boldsymbol{y}_K^i$
14 compute biased gradient $\tilde{\boldsymbol{g}}_t \leftarrow \mathcal{I}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{s}_t, K, \eta)$
15 update global optimiser values $\boldsymbol{s}_{t+1} \leftarrow \mathcal{T}(\tilde{\boldsymbol{g}}_t, \boldsymbol{s}_t)$
16 **end**

Participating clients download the global model $\boldsymbol{x}_t$ and optimiser values $\boldsymbol{s}_t$ (line 5), and then perform K steps of adaptive optimisation using the generic update step $\mathcal{U}$ and locally-computed stochastic gradients (lines 7-10). After local training, all participating clients upload their local models to the server (line 11). The server computes the new global model $\boldsymbol{x}_{t+1}$ as an average of client models (line 13). It performs the generic inverse-step $\mathcal{I}$ to compute the average gradient during the client local updates (line 14). The server then updates the global statistics using the generic tracking step $\mathcal{T}$ (line 15).

Table 6.1: Update, Tracking and Inverse steps for three popular optimisers with: decay rates β, β_1, β_2; (stochastic) gradients $\boldsymbol{g}$; stability constant ϵ; learning rate η and total local steps K; and previous and current global models $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$.

	Stats	Update ($\mathcal{U}$)	Tracking ($\mathcal{T}$)	Inverse ($\mathcal{I}$)
RMSProp	$\boldsymbol{v}$	$\frac{\boldsymbol{g}}{\sqrt{\boldsymbol{v}}+\epsilon}$	$\boldsymbol{v} \leftarrow \beta \boldsymbol{v} + (1-\beta)\boldsymbol{g}$	$\frac{1}{\eta K}(\boldsymbol{x}_t - \boldsymbol{x}_{t+1})(\sqrt{\boldsymbol{v}} + \epsilon)$
SGDm	$\boldsymbol{m}$	$\beta \boldsymbol{m} + (1-\beta)\boldsymbol{g}$	$\boldsymbol{m} \leftarrow \beta \boldsymbol{m} + (1-\beta)\boldsymbol{g}$	$\frac{1}{1-\beta}(\frac{\boldsymbol{x}_t - \boldsymbol{x}_{t+1}}{\eta K} - \beta \boldsymbol{m})$
Adam	$\boldsymbol{m}, \boldsymbol{v}$	$\frac{\beta_1 \boldsymbol{m} + (1-\beta_1)\boldsymbol{g}}{\sqrt{v}+\epsilon}$	$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\boldsymbol{g}$ $\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1-\beta_2)\boldsymbol{g}$	$\frac{1}{1-\beta_1}\left(\frac{(\boldsymbol{x}_t - \boldsymbol{x}_{t+1})(\sqrt{\boldsymbol{v}}+\epsilon)}{\eta K} - \beta_1 \boldsymbol{m}\right)$

An alternative implementation of FedGBO (that would not require optimisers to be represented using an inverse step $\mathcal{I}$) would update the local model $\boldsymbol{y}_k^i$ on clients, whilst also maintaining a copy of the average local gradient (which would be the same size as the model used, $|\boldsymbol{x}|$). This average local gradient would then be uploaded to the server, and could be used to update the global model and global statistics instead of performing an inverse step $\mathcal{I}$. However, FedGBO is designed to be lightweight in terms of computation and memory cost for clients, as real-world FL scenarios can involve low-powered clients such as IoT devices. Therefore, this design decreases the memory and total computational cost for clients as the average gradient does not have to be stored alongside the local model $\boldsymbol{y}_k^i$.

FedGBO shares some design similarities with other adaptive-FL algorithms. The design differences between FedGBO and these algorithms is described below.

- *AdaptiveFedOpt* [25]: clients download a global model each round and perform vanilla SGD locally. The server generates a 'psuedogradient' each round from client model uploads. This psuedogradient is fed into an adaptive optimiser that exists only on the server to update the global model at the end of each round.
- *MFL* [39]: clients download a global optimiser each round (like FedGBO). However, the optimiser values are allowed to change during the client loop (as opposed to fixed). The now-unique client optimisers are uploaded to the server each round along with the client models. The server averages the client optimisers to make the next round's global optimiser (as opposed to updating it via the $\mathcal{I}$ and $\mathcal{T}$ steps) alongside averaging the client models to produce the next global model.
- *Mimelite* [27]: clients download a global model and optimiser each round and keep the values fixed during the client loop (like in FedGBO). However, clients also compute a full-batch gradient ($\boldsymbol{\nabla} f$) before local training and send this gradient along with the model to the server after local training (FedGBO does not compute or send $\boldsymbol{\nabla} f$). The server uses $\boldsymbol{\nabla} f$ to update the global optimiser (as opposed to updating it via the $\mathcal{I}$ and $\mathcal{T}$ steps).

Table 6.2 summaries the practical differences between the algorithms. The beneficial aspects of FedGBO compared to the other algorithms are: compared to the vanilla SGD used by FedAvg, FedProx, FedMAX and AdaptiveFedOpt, FedGBO's client-side optimisation can substantially accelerate model convergence; compared to MFL, FedGBO accounts for client-drift with fixed optimiser values, which can help accelerate convergence and has lower upload cost depending on the optimiser used (50% less for SGDm/RMSProp, 66% less for Adam); compared to Mimelite, FedGBO does not require the expensive computation of $\boldsymbol{\nabla} f$ and has 50% reduced upload cost. Reducing upload costs in FL is of particular importance due to the asymmetric bandwidth of devices at the network edge.

Table 6.2: Popular state-of-the-art FL algorithms (plus FedAvg), with per-client download (Down) and upload (UP) costs and per-client memory requirements. $\boldsymbol{x}$ denotes the FL model, and $\boldsymbol{s}$ denotes the federated optimiser values.

Algorithm	Approach	Down	Up	Memory
FedAvg [24]	Averages models	$\boldsymbol{x}$	$\boldsymbol{x}$	$\mathcal{O}(\lvert\boldsymbol{x}\rvert)$
FedProx [36]	Proximal term	$\boldsymbol{x}$	$\boldsymbol{x}$	$\mathcal{O}(2\lvert\boldsymbol{x}\rvert)$
FedMAX [102]	Max-entropy term	$\boldsymbol{x}$	$\boldsymbol{x}$	$\mathcal{O}(\lvert\boldsymbol{x}\rvert)$
AdaptiveFedOpt [25]	Server-only optimiser	$\boldsymbol{x}$	$\boldsymbol{x}$	$\mathcal{O}(\lvert\boldsymbol{x}\rvert)$
MFL [39]	Averages models and optimisers	$\boldsymbol{x}, \boldsymbol{s}$	$\boldsymbol{x}, \boldsymbol{s}$	$\mathcal{O}(\lvert\boldsymbol{x}\rvert + \lvert\boldsymbol{s}\rvert)$
Mimelite [27]	Unbiased global optimiser	$\boldsymbol{x}, \boldsymbol{s}$	$\boldsymbol{x}, \boldsymbol{\nabla f}$	$\mathcal{O}(2\lvert\boldsymbol{x}\rvert + \lvert\boldsymbol{s}\rvert)$
FedGBO	Biased global optimiser	$\boldsymbol{x}, \boldsymbol{s}$	$\boldsymbol{x}$	$\mathcal{O}(\lvert\boldsymbol{x}\rvert + \lvert\boldsymbol{s}\rvert)$

6.1.2 Reducing Client Drift

FedGBO reduces client-drift by keeping a set of global statistics (downloaded at the start of each communication round) that are not updated in the local-training loop, as shown in Algorithm 6. For FedGBO with a momentum-based optimiser, when the 'decay' parameter (e.g., β in SGDm) is large, there is less influence from client gradients in the local updates, and more influence from the global biased statistics, so the models uploaded by clients will be more similar. For adaptive-learning rate methods (e.g., RMSProp), using the same fixed adaptive parameters on all clients

scales the local updates performed by clients for a given model parameter by the same value, as opposed to letting the adaptive parameters change during the local update.

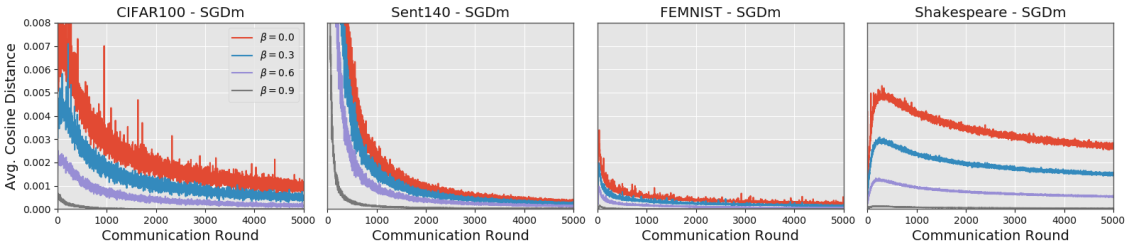


Figure 6.1: Average cosine distances between client models uploaded at the end of each communication round. Curves are averages over 5 random trials. Note FedGBO with $\beta = 0$ is equivalent to FedAvg.

Figure 5.1 plots the average cosine distance between client models at the end of each communication round, for four benchmark FL datasets (further details about datasets and models are given in Section 6.3), using FedGBO with the SGDm optimiser. When $\beta = 0$ (red curves), there is no influence from the global biased momentum, and FedGBO is equivalent to FedAvg. As explained above, Figure 5.1 (a) - (d) show that increasing the decay parameter (β) causes client updates to be more aligned due to more influence from the fixed global momentum. Figure 5.1 therefore demonstrates how FedGBO can tackle the problem of client-drift.

6.2 Convergence Analysis

In this section, it is shown that the updates to the global model in FedGBO with a generic optimiser can be formulated as updates to the same optimiser in the centralised setting, with a perturbed gradient and statistics update. This formulation can be used to extend existing convergence analyses, and is applied to a recent analysis of SGDm [103], recovering the same tight dependence on β as in the centralised analysis, with added client-drift terms associated with FL.

In FL, a model $\boldsymbol{x} \in \mathbb{R}^d$ is trained to minimise the following objective:

$$F(\boldsymbol{x}) = \mathop{\mathbb{E}}_{i}\left[f_i(\boldsymbol{x}) = \frac{1}{n_i}\sum_{j=1}^{n_i} f(\boldsymbol{x}; \xi_{i,j})\right], \tag{6.1}$$

where f_i, n_i and $\{\xi_{i,1}, \cdots, \xi_{i,n_i}\}$ denote the average loss, total number of samples, and training samples on client i, respectively. f is the loss function that clients use to train $\boldsymbol{x}$. Intuitively, the expected loss over all samples and all clients is minimised, as would be minimised by training in the centralised setting over pooled data. The individual client losses f_i are presented as a sum and the global loss F as an expectation to emphasise that in FL there are a very large number of clients, each possessing a small number of local samples.

Clients are assumed to have heterogeneous local data distributions. Therefore, the local minimiser f^* for any two clients i and j are not necessarily the same: $f_i^* \neq f_j^*$ (see Chapter 2). This is the source of client drift for $K > 1$ local updates. The following standard assumptions are made in the analysis of FedGBO.

Assumption 1 (Lower bound). *F is bounded below by F^*: $F(\boldsymbol{x}) > F^*, \forall \boldsymbol{x} \in \mathbb{R}^d$.*

Assumption 2 (Inter-client variance). *The variance of client gradients is bounded:* $\mathop{\mathbb{E}}_{i}[\|\nabla f_i(\boldsymbol{x}) - \nabla F(\boldsymbol{x})\|^2] \leq G^2, \forall\ i.$

Assumption 3 (Gradient magnitude). *The magnitude of client gradients is bounded:* $\|f_i(\boldsymbol{x}; \xi)\|^2 \leq R^2, \forall\ i.$

Assumption 4 (Lipschitz gradients). *Client loss functions are L-smooth:* $\|\nabla f_i(\boldsymbol{x}; \xi) - \nabla f_i(\boldsymbol{y}; \xi)\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall\ i$. *$F$ is a convex combination of f_i, so F is therefore also L-smooth.*

Assumption 5 (Intra-client variance). *Client stochastic gradients are unbiased estimates of the local gradients,* $\mathop{\mathbb{E}}_{\xi}[\nabla f_i(\boldsymbol{x}; \xi)] = \nabla f_i(\boldsymbol{x}), \forall\ i$, *with bounded variance:* $\mathop{\mathbb{E}}_{\xi}[\|\nabla f_i(\boldsymbol{x}; \xi) - \nabla f_i(\boldsymbol{x})\|^2] \leq \sigma^2, \forall\ i.$

Previously, [27] showed that FedAvg with $K > 1$ local updates could be reformulated as centralised optimisation with a perturbed gradient, which the authors then use to prove the convergence of their Mime algorithm. The perturbation $\boldsymbol{e}_t$ of the centralised-training gradient $\boldsymbol{g}_t$ is defined as:

$$\boldsymbol{e}_t = \frac{1}{K|S|} \sum_{i=1}^{K} \sum_{i \in S} \left(\nabla f_i(\boldsymbol{y}_{i,k-1}; \xi_{i,k}) - \nabla f_i(\boldsymbol{x}; \xi_{i,k}) \right), \tag{6.2}$$

for local iterations $\{y_{i,0}, \cdots, y_{i,K-1}\}$, and set of sampled clients S. Thus, the gradient-perturbation $\boldsymbol{e}_t$ is the average difference between client gradients (over the K local steps and S clients) and the gradients that would have been computed using the global model. Using this definition, the updates of FedGBO can be written as perturbed updates to a centralised optimiser:

$$\begin{aligned} \boldsymbol{x}_t &\leftarrow \boldsymbol{x}_{t-1} - \tilde{\eta} \mathcal{U}(\boldsymbol{g}_t + \boldsymbol{e}_t, \boldsymbol{s}_{t-1}), \\ \boldsymbol{s}_t &\leftarrow \mathcal{V}(\boldsymbol{g}_t + \boldsymbol{e}_t, \boldsymbol{s}_{t-1}), \end{aligned}$$

for generic model-update step $\mathcal{U}$, optimiser tracking step $\mathcal{T}$, and 'pseudo-learning-rate' $\tilde{\eta} = K\eta$. The local updates of FedGBO use fixed global statistics in a similar manner to the fixed statistics used in Mime (albeit we use biased global statistics). As the above assumptions are at least as strong as those from [27], their result can be used to bound the norm of the perturbation $\boldsymbol{e}_t$:

$$\mathop{\mathbb{E}}_{t}\left[\|\boldsymbol{e}_t\|^2\right] \leq (B^2 L^2 \tilde{\eta}^2) \left(\mathop{\mathbb{E}}_{t}\left[\|\boldsymbol{g}_t\|^2\right] + G^2 + \frac{\sigma^2}{K} \right), \tag{6.3}$$

where $B \geq 0$ is a bound on the Lipschitzness of the optimiser $\mathcal{U}$'s update: $\|\mathcal{U}(\boldsymbol{g})\| \leq B\|\boldsymbol{g}\|$.

Using the perturbed gradient and momentum generalisation presented above, updates of FedGBO using the SGDm optimiser can be written as:

$$\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + (\boldsymbol{g}_{t-1} + \boldsymbol{e}_{t-1}) \tag{6.4}$$

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \tilde{\eta} \boldsymbol{m}_t. \tag{6.5}$$

While the specific SGDm implementation used in the later experiments multiplies the second term of (6.4) by a $(1 - \beta)$ term (hyperparameter choice with this implementation tends to be more intuitive), the $(1-\beta)$ term in (6.4) is dropped to simplify

the proof, and it is assumed the $(1-\beta)$ and β terms can be incorporated into $\tilde{\eta}$. As is typical for nonconvex analysis, the expected squared norm of the model gradient (at communication round t) is bounded. For total iterations T, define t as a random index with value $\{0, \cdots, T-1\}$, with probability distribution $\mathbb{P}[t=j] \propto 1-\beta^{T-j}$. Full proofs are given in Appendix B.

Theorem 6.1 (Convergence of FedGBO using SGDm) *Let the assumptions A1-A5 hold, the total number of iterations $T > (1-\beta)^{-1}$, $\tilde{\eta} = K\eta > 0, 0 \leq \beta < 1$, and using the update steps given by (6.4) and (6.5):*

$$\mathbb{E}[\|\nabla F(\boldsymbol{x}_t)\|^2] \leq \frac{2(1-\beta)}{\tilde{\eta}\tilde{T}}(F(\boldsymbol{x}_0) - F^*) + \frac{T\tilde{\eta}^2 Y}{\tilde{T}} + \frac{2T\tilde{\eta}L(1+\beta)(R^2+\tilde{\eta}^2 Y)}{\tilde{T}(1-\beta)^2}, \quad (6.6)$$

where $\tilde{T} = T - \frac{\beta}{1-\beta}$, and $Y = 18B^2L^2\left(R^2 + G^2 + \frac{\sigma^2}{K}\right)$.

Theorem 6.1 therefore shows that the perturbed gradient and momentum framework presented above can be used to apply an existing convergence proof from a centralised optimiser to FedGBO. Next, Theorem 6.1 is simplified to achieve a more explicit convergence rate.

Corollary 6.1 (Theorem 6.1 simplification using iteration assumption) *Making the same assumptions as in Theorem 1, and also that $T \gg (1-\beta)^{-1}$, hence $\tilde{T} \approx T$, and using $\tilde{\eta} = (1-\beta)\frac{2C}{\sqrt{T}}$, for $C > 0$, then:*

$$\mathbb{E}[\|\nabla F(\boldsymbol{x}_t)\|^2] \leq \frac{F(\boldsymbol{x}_0) - F^*}{C\sqrt{T}} + \frac{4CL(1+\beta)(R^2 + Z/T)}{(1-\beta)\sqrt{T}} + \frac{Z}{T}, \quad (6.7)$$

where $Z = 4C^2(1-\beta)^2 Y$.

Corollary 6.1 shows that the gradients of FedGBO can be bounded for nonconvex objectives. This is the first convergence analysis of an FL-algorithm using SGDm for nonconvex objectives: [39] analyses strongly-convex objectives with deterministic gradients, and [25, 27] analyse Adam with $\beta_1 = 0$ (i.e., RMSProp). The lack of directly-related analyses makes comparison of this analysis to existing works less clear, but Corollary 6.1 still provides useful insights into FedGBO's convergence.

- **Relation to centralised rate:** Comparing (6.7) to Theorem B.1 of [103] shows that the $\mathcal{O}(1/\sqrt{T})$ convergence rate can be retained (with the same state-of-the-art $(1-\beta)$ denominator). However, (6.7) contains added error terms due to client-drift: Z in the second term arises from the biased momentum used in FedGBO, and the third term arises from biased client gradients.

- **Setting $\tilde{\eta}$:** Scaling $\tilde{\eta}$ with $\mathcal{O}(1/\sqrt{T})$ fortunately decreases the error due to biased client gradients and momentum with $\mathcal{O}(1/T)$ and $\mathcal{O}(1/\sqrt{T})$, respectively. As $\tilde{\eta} = K\eta$, this indicates that either K or η could be decayed during training to balance fast convergence and final error.

- **Effect of K:** According to (6.7), there is no overt benefit to performing $K > 1$ local steps, which is a common theme within FL analysis. [39] proves the convergence of MFL by bounding the distance to centralised momentum-SGD, and this distance naturally converges to 0 as $K \to 1$. Similarly, [26, 27] improve convergence rates for convex objectives when $K > 1$ by adding Variance-Reduction (VR)

to the client objective (which increases communication and computation costs). Without VR, the convergence of their algorithms do not dominate distributed SGD (with $K = 1$). Despite the lack of theoretical benefit from $K > 1$, the experimental studies of other works and Section 6.3 shows large K can significantly improve convergence rate empirically.

Table 6.3: Details of learning tasks used in experiments.

Task	Type	Classes	Model	Total Clients	Clients per Round	Samples per Client
CIFAR100	Image classification	100	CNN	500	5	100
Sent140	Sentiment analysis	2	Linear	21876	22	15
FEMNIST	Image classification	62	CNN	3000	30	170
Shakespeare	Next-character prediction	79	GRU	660	7	5573

6.3 Experiments

In this section, a comprehensive comparison between FedGBO and various state-of-the-art adaptive-FL algorithms (AdaptiveFedOpt [25], MFL [39], Mimelite [27]), and non-adaptive algorithms (FedAvg [24], FedProx [36], FedMAX [102]) is conducted. The experiments use 4 benchmark FL datasets, from the domains of computer vision, sentiment analysis, and language modelling. Each adaptive-FL algorithm is compatible with a variety of optimisers, so three of the most popular and ubiquitous are tested: RMSProp [101], SGDm [100], and Adam [87]. Although from an algorithmic perspective RMSProp and SGDm are simply special cases of Adam (with $\beta_1 = 0$ and $\beta_2 = 0$, respectively), for FL there is the important practical difference of not requiring the zeroed parameters to be communicated. All of the models and algorithms tested were implemented with PyTorch 1.7.0, and run on workstations equipped with Intel i9 CPUs and NVidia RTX 3090 GPUs, running Ubuntu 20.04. Table 6.3 gives a brief overview of the datasets and models used, with more thorough details below.

CIFAR100: a federated version of the CIFAR100 dataset consisting of (32×32) pixel RGB images from 100 classes. Training samples are partitioned according to the class labels into 500 workers using the Pachinko Allocation Method first used in [25]. Like in similar FL works [25, 27], pre-processing was applied to the training samples comprising a random horizontal flip ($p = 0.5$) followed by a random crop of the (28×28) pixel sub-image. A CNN with the following architecture was trained: a $(3 \times 3 \times 32)$ ReLU Convolution layer, (2×2) max pooling, $(3 \times 3 \times 64)$ ReLU Convolution layer, (2×2) max pooling, a 512-unit ReLU fully connected layer, and softmax output, with batch size of 32.

Sent140: a sentiment analysis task using Twitter posts, pre-processed using the LEAF FL benchmark suite [99]. Tweets are grouped by user, users with < 10 posts are discarded, and 20% of each users' samples are taken for the test set. A normalised bag-of-words vector of length 5k (representing presence of the 5k most common token in the dataset) was created for each samples, with each target being a vector of length two (positive or negative sentiment). Samples without any of the top 5k token were discarded. A linear model using batch size 8 was trained. Note

that FedMAX is not trained on Sent140 as the algorithm is only compatible with DNNs.

FEMNIST: a federated version of the EMNIST dataset containing (28×28) pixel greyscale images from 62 classes, pre-processed using LEAF [99]. Samples are grouped into users by the writer of the symbol, and 20% of each user's samples are grouped to make a test-set. 3000 clients (from the maximum 3550) were selected for use in all experiments. A CNN with the same architecture as the CIFAR100 model was trained (albeit with 62 outputs) using a batch size of 32.

Shakespeare: a next-character prediction task of the complete plays of William Shakespeare, pre-processed using LEAF [99]. The lines from all plays are grouped by speaker (e.g., Macbeth, Lady Macbeth etc.). Speakers with < 2 lines are discarded, leaving 660 total clients. Lines are processed into sequences of length 80, with a vocabulary size of 79, and the last 20% of each client's lines are taken to produce the test-set. A GRU model was trained on the dataset comprising a trained embedding layer with 8 outputs, two stacked GRU layers with 128 outputs each, and a softmax output layer, using a batch size of 32.

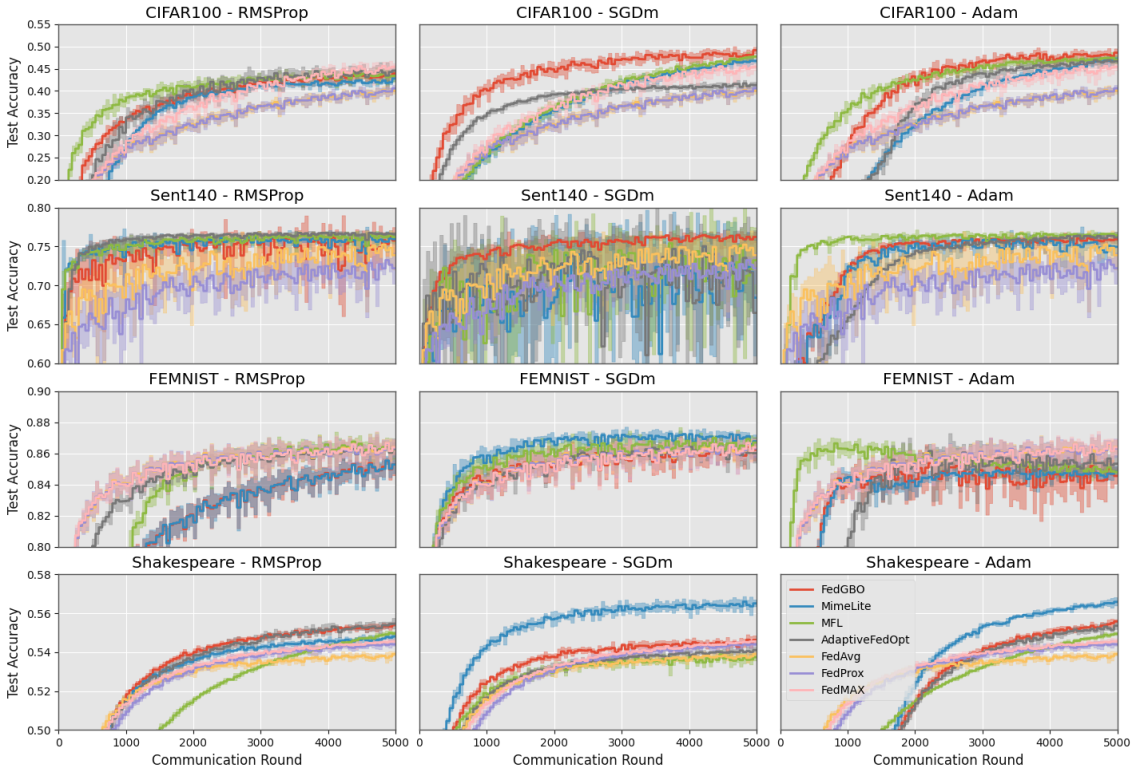


Figure 6.2: Comparison of adaptive FL algorithms on different FL datasets, using $K = 10$ local update steps. Lines show average over 5 random trials, shaded regions show 95% confidence intervals of the mean.

6.3.1 Convergence Speed

Convergence speeds of the different adaptive-FL algorithms using each optimiser on each dataset are first compared. For each [dataset, FL algorithm, optimiser] combination, hyperparameters were tuned via grid search in order to achieve the highest test-set accuracy within 5k communication rounds (evaluated every 50 rounds).

Learning rates were tested in the range $\eta \in [10^1, 10^{-5}]$, with step size of 0.1, and optimiser parameter $\beta = \{0.3, 0.6, 0.9, 0.99, 0.999, 0.9999\}$. For Adam, in order to keep the number of experiments performed feasible, $\beta_2 = 0.99$ was fixed for all experiments and β_1 was tuned as above. For RMSProp and Adam, as has been noted previously [25, 27] it was found that a large adaptivity parameter $\epsilon = 10^{-3}$ was required for stable convergence. Tuning AdaptiveFedOpt took significantly more simulations compared to the other algorithms due to the extra hyperparameter (the server learning rate, on top of client learning rate and optimiser parameter), which represents a drawback for real-world use.

For FedAvg only η was tuned. For FedProx η and proximal term μ were tuned. For FedMAX η and entropy-loss-weighting α was tuned. These algorithms do not use adaptive optimisation, so the same curve for each algorithm in each row of Figures 6.2 and 6.3 are presented. Well over 1000 simulations were conducted in total.

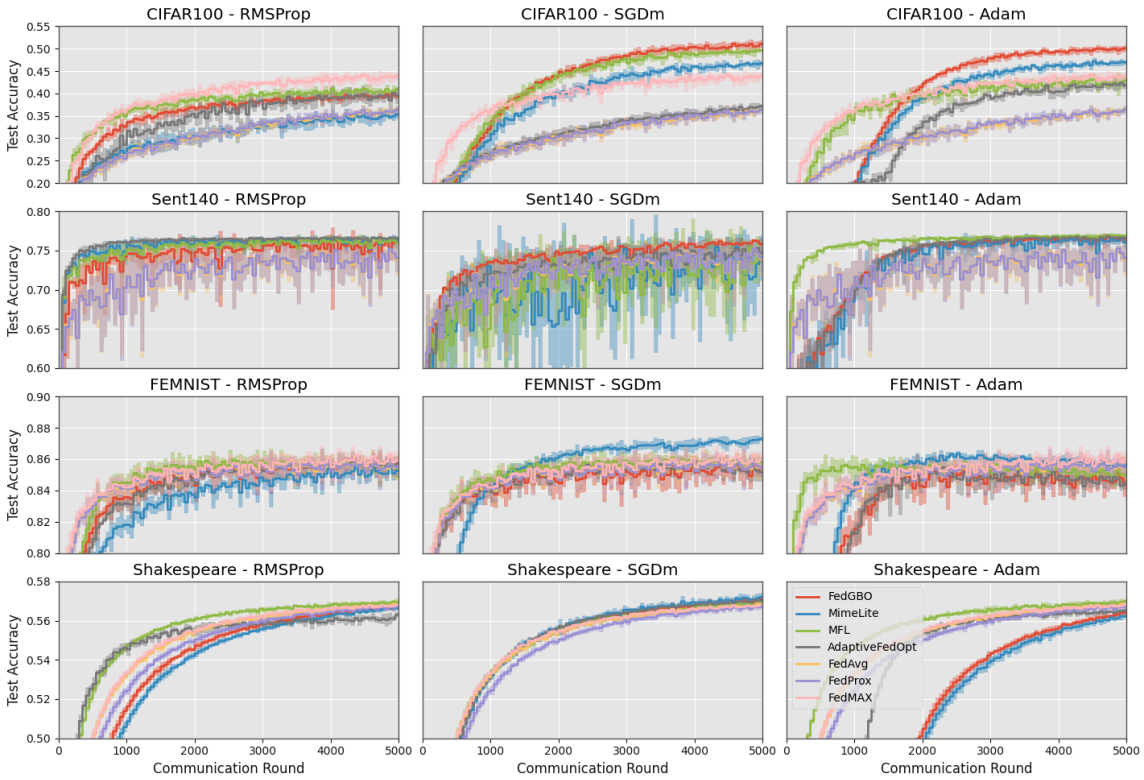


Figure 6.3: Comparison of adaptive-FL algorithms on different FL datasets, using $K = 50$ local update steps. Lines show average over 5 random trials, shaded regions show 95% confidence intervals of the mean.

In Figure 6.2, the test-set accuracies during training for these tuned parameters using $K = 10$ local steps are presented. This a 'low-K' regime, representing scenarios where clients can communicate more frequently with the server. Figure 3 shows the accuracy curves for a 'high-K' ($K = 50$) regime. The practical insights provided by these experiments are discussed below.

Choice of FL algorithm: Comparing the rows of Figures 6.2 and 6.3, there is no clear FL algorithm that has universally superior performance in terms of convergence speed. For CIFAR100 and $K = 10$, FedGBO with SGDm converges fastest, whereas for the Shakespeare, $K = 10$, Mimelite with SGDm converges fastest. Furthermore, for FEMNIST none of the tested algorithms provided significant benefit compared

to FedAvg in both the $K = 10$ and $K = 50$ scenarios. The reason for this result is likely because FEMNIST is a relatively simple greyscale task, so more sophisticated optimisation techniques do not much help convergence.

Therefore, when planning a real-world FL deployment, the potential benefit of adaptive-FL should be taken into account. If the learning task is relatively simple, additional hyperparameters (which then need to be tuned) and communicated data added by adaptive-FL algorithms may outweigh the performance gain. Some non-adaptive FL algorithms can show enhanced performance compared to FedAvg without adding to the communication or computation cost (e.g., FedMAX for CIFAR100). However, on the more challenging CIFAR100 and Shakespeare tasks, adaptive FL algorithms provide significant speedup compared to FedAvg.

Comparing Figures 6.2 and 6.3, some datasets and algorithms benefited from increased local computation, whereas others were hindered. For CIFAR100 with RMSProp, the best accuracy that all algorithms could achieve was lowered for $K = 50$. For Shakespeare, almost all algorithms and optimisers were able to converge faster for $K = 50$, and the ordering of which algorithm achieved the best accuracy changed. The relative performance of adaptive-FL algorithms on the different benchmark datasets warrants further investigation in future works, but task complexity and number of local samples appear to be important factors.

Alongside this, the communication costs of the adaptive-FL algorithms should be taken into account. A more detailed analysis of the results considering final model accuracy, upload cost, and total computation is presented in Section 6.3.2.

Choice of optimiser: As shown in Figure 6.2, there was also no universally-best optimiser for each dataset and adaptive-FL algorithm, which matches the finding for centralised training [104]. For example, the fastest convergence for Shakespeare, $K = 10$ was using Mimelite with SGDm, whereas for Sent140 it was RMSProp and Adam in both $K = 10$ and $K = 50$ scenarios.

In both scenarios for Sent140, SGDm provided little improvement compared to FedAvg. The gradients computed in the Sent140 task are sparse (due to very sparse features). Adaptive learning-rate methods like RMSProp and Adam are known to provide good convergence rates for sparse-gradient tasks [105], whereas the very noisy convergence of SGDm is due to the learning rate having to be set very large for this task ($\eta = 10.0$). For the $K = 50$ scenario, the learning rate could be set lower, resulting in slightly less erratic convergence and narrower confidence intervals.

On the other hand, the rest of the tasks do not have sparse features, and RMSProp performed largely the worst. Therefore, these results suggest that when choosing an adaptive-FL optimiser, a good initial choice of optimiser is the one that works best on the task in the standard centralised setting. Also, as shown in Table 6.2, optimiser choice impacts the communication cost of FedGBO, MFL and Mimelite: for RMSProp and SGDm, the total download cost is doubled, and tripled for Adam. The relationship between convergence speed gain due to optimiser choice and increased communication cost represents another design consideration for FL engineers.

Combining strategies: The adaptive-FL algorithms studied (FedGBO, Mimelite, MFL, AdaptiveFL) can improve the convergence rate of the global model through adaptive optimisation. However, in some cases simply modifying the local objective can result in faster convergence (e.g., FedMAX on CIFAR100). The adaptive-FL algorithms could readily be combined with objective-modifying algorithms, which may

provide even greater performance without additional overheads. This combination presents an interesting avenue for future research.

6.3.2 Considering Convergence, Communication and Computation

There are many factors to consider when designing an FL algorithm for real-world use. Alongside convergence speed, the total computational cost of FL is very important. In cross-device FL, clients may be a highly diverse set of devices with a wide range of computational power (e.g., different generations of smartphones) [7]. This has two practical implications: the time taken to perform local training may be the most significant bottleneck [97], and FL deployments that drop stragglers [8] may end up regularly dropping the same clients, skewing the global model in favour of faster clients. Furthermore, research indicates the total energy consumed by FL can exceed the energy consumed by centralised training, motivating FL algorithms with lower computational costs [35].

Table 6.4: Experimental results for different algorithms. The left section displays the maximum accuracy achieved by each algorithm (95% confidence intervals in brackets), and the total upload cost and FLOPs to achieve it. The right section shows the number of rounds taken to match FedAvg's accuracy, with the corresponding upload cost and FLOPs. Adaptive-FL algorithms use SGDm.

Algorithm	Max Acc (%)	Upload (GB)	FLOPs ($\times 10^{12}$)	Upload (GB)	FLOPs ($\times 10^{12}$)
CIFAR100					
FedAvg	40.2 (±0.9)	112	120	-	-
FedProx	40.6 (±0.7)	115	124	115	124
FedMAX	45.6 (±1.0)	**112**	**120**	65.2	69.7
AFO	41.7 (±0.8)	112	120	59.4	63.5
MFL	47.7 (±0.3)	231	125	126	68.3
Mimelite	46.8 (±0.4)	231	163	130	92.0
FedGBO	**49.2** (±0.7)	115	124	**22.1**	**23.8**
Sent140					
FedAvg	75.2 (±0.4)	3.9	0.27	-	-
FedProx	73.3 (±0.4)	4.0	0.31	-	-
AFO	75.0 (±1.0)	**3.4**	**0.24**	-	-
MFL	75.4 (±0.8)	6.1	0.26	6.1	0.26
Mimelite	**75.0** (±2.0)	6.9	0.32	-	-
FedGBO	**76.5** (±0.2)	**3.4**	0.27	**1.2**	**0.09**
FEMNIST					
FedAvg	86.6 (±0.2)	**423**	**416**	-	-
FedProx	86.6 (±0.2)	472	469	423	420
FedMAX	86.7 (±0.2)	472	464	423	416
AFO	86.4 (±0.6)	433	426	389	383
MFL	**87.0** (±0.1)	846	423	418	209
Mimelite	**87.3** (±0.2)	720	544	**282**	**213**
FedGBO	86.4 (±0.3)	447	444	447	444
Shakespeare					
FedAvg	53.9 (±0.1)	21.0	409	-	-
FedProx	54.4 (±0.1)	**20.4**	396	13.0	396
FedMAX	54.6 (±0.1)	21.0	409	12.1	235
AFO	54.1 (±0.1)	21.0	409	16.1	314
MFL	53.9 (±0.2)	37.8	**368**	-	-
Mimelite	**56.6** (±0.3)	40.8	7325	**7.6**	1373
FedGBO	54.6 (±0.3)	20.6	401	8.9	**173**

As well as model convergence and computation cost, other important factors include the total data communicated during FL (especially uploaded data considering the asymmetric bandwidth of the network edge), number of algorithm hyperparameters, and more. It is therefore difficult to compare FL algorithms when considering all these factors. In Table 6.4, the tested algorithms are summarised using several factors by combining the per-round communication costs from Table 6.2, the convergence results from Figure 6.2 and the computation costs of each algorithm for the $K = 10$ scenario using SGDm. Details about calculating the computation costs of the FL algorithms are given in Appendix B.

The left-hand section of Table 6.4 shows the maximum accuracy of each algorithm, and the total upload cost and FLOPs (Floating Point Operations) required to reach it (number of rounds multiplied by per-round upload/FLOPs). The right-hand section shows the total data and FLOPs required to match the maximum FedAvg accuracy for that scenario, giving an indication of convergence speed.

Table 6.4 shows that for CIFAR100, FedGBO achieved the highest accuracy whilst also having near-lowest uploaded data and total FLOPs. To match FedAvg's accuracy, FedGBO had by far the lowest upload cost and FLOPs, much lower than FedAvg. Similar performance is shown with Sent140. For FEMNIST, FedGBO had competitive performance in terms of model accuracy, upload cost and FLOPs, and for Shakespeare, had competitive accuracy whilst having among the lowest upload cost and computation. FedGBO therefore shows a good trade-off between convergence rate, final accuracy, uploaded data, and computational cost.

Mimelite and FedGBO both use fixed global optimisers. However, the computation performed by Mimelite is much higher than FedGBO due to computing full-batch gradients (see Appendix B). This motivates the question of whether computing these full-batch gradients is a good use of local computation, especially considering the results from the Shakespeare scenario (where Mimelite achieved the highest model accuracy but with more than 10× the total computation). To test this, Mimelite was modified to use minibatch unbiased gradients instead (denoted 'MimeXlite': Mime-'extra'-light). MimeXlite has the same computational cost as FedGBO, but 2× the upload cost.

Table 6.5: Maximum accuracy achieved by FedGBO, Mimelite and MimeXlite on the Shakespeare dataset (95% confidence intervals given in brackets).

	Dataset	FedGBO	Mimelite	MimeXlite
K = 10	RMSProp	55.4 (±0.1)	54.8 (±0.2)	55.9 (±0.3)
	SGDm	54.6 (±0.3)	56.6 (±0.3)	51.9 (±0.3)
	Adam	55.6 (±0.1)	56.6 (±0.2)	52.8 (±0.2)
K = 50	RMSProp	56.8 (±0.1)	56.7 (±0.2)	55.9 (±0.1)
	SGDm	57.1 (±0.1)	57.2 (±0.2)	50.4 (±0.3)
	Adam	56.4 (±0.2)	56.2 (±0.1)	52.5 (±0.2)

Table 6.5 shows that the performance of MimeXlite is significantly worse than FedGBO or Mimelite in almost all Shakespeare scenarios. The performance drop is worst for SGDm and Adam. This is likely due to the high variance of minibatch gradients harming the momentum parameters (as used in SGDm and Adam), but having a lesser impact on RMSProp. FedGBO may 'get away' with computing only minibatch gradients because the optimiser gradients are averaged over the local steps, helping to lower the variance of the global optimiser gradients (despite these gradients being biased).

6.4 Chapter Summary

In this Chapter, the Federated Global Biased Optimiser (FedGBO) algorithm was proposed, incorporating a set of global statistics for a generic machine learning optimiser within FedAvg. FedGBO demonstrates fast convergence rates whilst having lower communication and computation costs compared to other state-of-the-art FL algorithms that use adaptive optimisation. It was showed that FedGBO with a generic optimiser can be formulated as centralised training using a perturbed gradient and optimiser update, allowing analysis of generic centralised optimisers to be extended to FedGBO. The same convergence rate for FedGBO with SGDm was achieved as in a recent analysis on centralised SGDm, plus an extra FL-related error term that decays with η^2. An extensive set of comparison experiments were performed using 6 competing FL algorithms (3 of which also use adaptive optimisation) and 3 different optimisers on 4 benchmark FL datasets. These experiments highlighted FedGBO's highly competitive performance, especially considering FedGBO's low computation and communication costs. Practical insights into the choice of adaptive-FL algorithms and optimisers, and communication and computation trade-offs within FL were also provided based on the experimental results.

Chapter 7

Conclusion & Future Work

This chapter concludes the thesis by providing first a short summary of contributions, followed by an overview of current limitations to the state-of-the-art and potential avenues for future research.

7.1 Thesis Summary

Federated Learning (FL) is a recent paradigm within Machine Learning (ML) for creating ML models from user data in a distributed fashion, without users sharing their training data with any other party. FL has received huge research interest from academia and industry due to its potential for privacy-preserving ML. Chapters 3 - 6 presented novel algorithms and theoretical contributions related to FL, with the broad aims of reducing computation and network-communication costs, improving the generalisation of FL models across heterogeneous client datasets, and accelerating model-convergence rates.

Most FL deployments consider a large number of users connected over the internet, meaning that network communication can be a major training bottleneck. The basic approach for training FL models in a communication-efficient manner is performing rounds of model training on clients followed by model communication and averaging. Chapter 3 presented a new algorithm that adds adaptive optimisation in order to improve FL's per-round convergence rate. Adaptive optimisation increases communication overheads, so novel compression techniques tailored to the adaptive optimisation algorithm were also proposed. This combination of techniques was named Communication-Efficient Federated Averaging (CE-FedAvg). Thorough simulations and testbed results showed that CE-FedAvg can simultaneously reduce the total communication cost and runtime of training compared to baselines, making FL more practicable for use in Edge Computing (EC).

In real-world FL, the data possessed by users is highly heterogeneous due to different behaviours and environments. The goal of Personalised-FL is to train unique models tailored to each user, rather than a single FL model. This may be more useful for deployments where the model(s) are used for on-device inference. Chapter 4 proposes a new Personalised-FL algorithm, Multi-Task Federated Learning (MTFL), which takes inspiration from the topic of multi-task learning to train unique models for each FL user, facilitated by model layers that are kept private rather than averaged along with the rest of the FL model. Theoretical analysis suggests that these personal layers help client models to match the output distributions that would be produced by purely local training, and extensive experiments show that MTFL can improve average model performance and reduce the training burden compared to FL and other Personalised-FL algorithms.

Performing more local computation in each communication round typically increases the convergence rate of FL and reduces the total communicated data to reach a given error. However, it usually comes at the cost of increased total computation, and with diminishing returns compared to centralised training. In Chapter 5, the principle of decaying the amount of work performed by users during FL was explored. The optimal number of Stochastic Gradient Descent steps performed by clients each round to maximise the convergence rate was derived for strongly-convex objectives. Simulated results using representative compute times and communication rates for the wireless EC scenario show that the total runtime of FL can be reduced whilst also significantly decreasing the total compute cost.

Many works have also proposed methods for incorporating adaptive optimisation into FL, however these approaches typically lead to higher computation, memory and communication costs compared to simpler algorithms. This limits their practicality for FL in EC using low-powered devices. Chapter 6 proposed the Federated Global Biased Optimiser (FedGBO) algorithm, which accelerates training whilst having lower communication, computation and memory overheads compared to other adaptive-FL algorithms. This is achieved by using adaptive optimiser statistics that are kept constant during the local update loop, which also helps to mitigate client-drift due to heterogeneous client data. FedGBO's convergence on nonconvex objectives was proven, and its empirical performance was demonstrated in a thorough experimental comparison on four benchmark datasets using three different adaptive optimisers.

7.2 Limitations & Future Work

As FL is a very recent sub-field within ML, there are a lack of realistic benchmarks that reflect the heterogeneous distribution of data across large-scale sets of users. Early research including the work in Chapters 3 and 4 took existing popular datasets such as CIFAR and designed artificial partitions in an attempt to reflect realistic user data [24]. However, there were no standardised partitioning methods, meaning the performance of algorithms was difficult to compare, and could be inflated by choosing splits amenable to the proposed algorithms [61]. Some attempts have been made to produce representative datasets [99], however many of these still create artificial partitions of existing datasets, and have yet to be widely adopted. To address this issue, future works should collect, pre-process and freely publish real-world data from a variety of FL tasks to make realistic benchmarks that are attractive to researchers. Datasets that exhibit concept-drift (changing distributions over time) would also be beneficial to the research community.

Similarly, analytics of real-world FL deployments would be very beneficial for researchers. A few technical papers of some deployments have been published [8] which provide insights that can be used to steer future research. These insights may cover aspects such as varying client participation rates over time, real-world systems-engineering considerations and directions for novel and useful applications and problems domains. Businesses may be reluctant to give precise details of deployments for commercial purposes, but inspiring relevant research will help improve commercial products in the long run.

Furthermore, due to the extremely high rate at which FL papers are being published, thorough and impartial comparison works covering multiple algorithms would be useful for researchers and industry. These survey papers would be aided by the realistic benchmarks and analytics suggested in the previous paragraphs. Alongside

comparing individual algorithms, combining elements of different algorithms could yield novel insights and findings. For example, many FL optimisation algorithms have been proposed with the aim of increasing per-round convergence rates, whilst many compression algorithms aim to reduce communication overheads at the cost of increased convergence rounds. Testing different optimisation and compression algorithms together (similar to the combined approach proposed in Chapter 3) would yield useful Pareto-optimal solutions on the communication-convergence trade-off, as has been done for convergence and final accuracy of FL models on convex objectives [30].

As is the case with much of ML, the theoretical progress in FL currently lags behind the empirical developments. Multiple works have proven that popular FL algorithms such as FedAvg can converge to a minimum point of convex [28] and nonconvex [26] objectives, and Chapter 5 proved the benefit of decaying the local computation over time for strongly-convex objectives. However, in order to prove dominant iteration complexity compared to simple distributed SGD, previous works have made additional assumptions or altered client objectives functions, leading to worse empirical performance or significantly increased computation and communication [36, 27, 76]. Further work is needed to tighten the converge bounds for popular and basic algorithms on nonconvex objectives, or to develop new algorithms with superior theoretical and empirical convergence with low costs.

Whilst FL provides the guarantee that sensitive data does not leave user devices during training, FL can still be vulnerable to information leakage. For example, gradient inversion attacks can be used to recreate user data and infer identity by malicious clients [106] or the server [107]. The MTFL algorithm proposed in Chapter 4 may help to mitigate this problem by reducing the total number of parameters shared by users, but still does not provide a formal guarantee. Technologies such as differential privacy and homomorphic encryption can be used to prevent these problems, but their study within FL has been limited. Their incorporation into popular FL algorithms should be thoroughly tested and theoretically analysed to understand the fundamental trade-off between privacy and utility within FL.

Bibliography

[1] D. Zhang, N. Maslej, E. Brynjolfsson, J. Etchemendy, T. Lyons, J. Manyika, H. Ngo, J. Niebles, M. Sellitto, E. Sakhaee, Y. Shoham, J. Clark, and R. Perrault, "The ai index 2022 annual report," p. 19, 2022.

[2] M. Hasan, "State of iot 2022: Number of connected iot devices growing 1814.4 billion globally," 2022. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[3] D. Reinsel, J. Gantz, and J. Ryding, "The digitization of the world from edge to core," *International Data Corporation (IDC)*, vol. 16, 2018.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, pp. 216–222, 2018.

[6] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[7] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[8] K. Bonawitz, H. Eichner, W. Grieskamp *et al.*, "Towards federated learning at scale: System design," in *Proc. Systems and Machine Learning Conference (SysML)*, 2019.

[9] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6341–6345, 2019.

[10] F. Hartmann, "Federated learning," Ph.D. dissertation, Free University of Berlin, 2018.

[11] T. M. Deist, F. J. Dankers, P. Ojha, M. S. Marshall, T. Janssen, C. Faivre-Finn, C. Masciocchi, V. Valentini, J. Wang, J. Chen *et al.*, "Distributed learning on 20 000+ lung cancer patients – the personal health train," *Radiotherapy and Oncology*, vol. 144, pp. 189–200, 2020.

[12] G. Shifmann, J. Zarate, N. Deshpande, R. Yeluri, and P. Peiravi, "Federated learning through revolutionary technology," 2021.

[13] Apheris, "Collaborative data ecosystems in manufacturing," 2022. [Online]. Available: https://www.apheris.com/federated-learning-platform-manufacturing

[14] "Federated learning on non-iid data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, 2021.

[15] L. Hao, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[16] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on Optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[18] "United kingdom mobile network experience report," Open Signal, Tech. Rep., 9 2021, accessed 01/08/2022. [Online]. Available: https://www.opensignal.com/reports/2021/09/uk/mobile-network-experience

[19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, and P.-l. Cantin, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, p. 1–12.

[20] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[21] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[22] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2022.

[23] S. Stich, "Local sgd converges fast and communicates little," in *International Conference on Learning Representations (ICLR)*, 2018.

[24] B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *International Conference on Artifical Intelligence and Statistics (AISTATS)*, pp. 1273–1282, 2017.

[25] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *Proc. International Conference on Learning Representations (ICLR)*, 2021.

[26] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. International Conference on Machine Learning (ICML)*, vol. 119, 2020, pp. 5132–5143.

[27] S. P. Karimireddy, M. Jaggi, S. Kale, M. Mohri, S. Reddi, S. U. Stich, and A. T. Suresh, "Breaking the centralized barrier for cross-device federated learning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 663–28 676, 2021.

[28] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations (ICLR)*, 2020.

[29] G. Malinovsky, K. Mishchenko, and P. Richtárik, "Server-side stepsizes and sampling without replacement provably help in federated optimization," *arXiv preprint: 2201.11066*, 2022.

[30] Z. Charles and J. Konečný, "Convergence and accuracy trade-offs in federated learning and meta-learning," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 130, 2021, pp. 2575–2583.

[31] J. Wang, R. Das, G. Joshi, S. Kale, Z. Xu, and T. Zhang, "On the unreasonable effectiveness of federated averaging with heterogeneous data," *arXiv preprint: 2206.04723*, 2022.

[32] G. Malinovskiy, D. Kovalev, E. Gasanov, L. Condat, and P. Richtarik, "From local SGD to local fixed-point methods for federated learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 119, 2020, pp. 6692–6701.

[33] H. Yang, M. Fang, and J. Liu, "Achieving linear speedup with partial worker participation in non-IID federated learning," in *International Conference on Learning Representations (ICLR)*, 2021.

[34] Z. Qu, K. Lin, Z. Li, J. Zhou, and Z. Zhou, "A unified linear speedup analysis of stochastic fedavg and nesterov accelerated fedavg," 2020.

[35] X. Qiu, T. Parcollet, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, "Can federated learning save the planet?" in *NeurIPS - Tackling Climate Change with Machine Learning*, 2020.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Machine Learning and Systems (MLSys)*, vol. 2, pp. 429–450, 2020.

[37] R. Pathak and M. J. Wainwright, "FedSplit: an algorithmic framework for fast federated optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7057–7066, 2020.

[38] C. T. Dinh, N. H. Tran, M. N. H. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 398–409, 2021.

[39] W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating Federated Learning via Momentum Gradient Descent," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1754–1766, aug 2020.

[40] P. Khanduri, P. Sharma, H. Yang, M. Hong, J. Liu, K. Rajawat, and P. K. Varshney, "Achieving optimal sample and communication complexities for non-iid federated learning," in *ICML Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2021.

[41] J. Wang, Z. Xu, Z. Garrett, Z. Charles, L. Liu, and G. Joshi, "Local adaptivity in federated learning: Convergence and consistency," *arXiv preprint arXiv:2106.02305*, 2021.

[42] R. Das, A. Acharya, A. Hashemi, S. Sanghavi, I. S. Dhillon, and U. Topcu, "Faster non-convex federated learning via global and local momentum," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, vol. 180, 2022, pp. 496–506.

[43] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[44] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," *2019 International Joint Conference on Neural Networks*, pp. 1–8, 2018.

[45] F. Sattler, S. Wiedemann, K. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[46] D. Neumann, F. Sattler, H. Kirchhoffer, S. Wiedemann, K. Müller, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek, "Deepcabac: Plug & play compression of neural network weights and weight updates," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 21–25.

[47] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary compression for communication-efficient federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 1162–1176, 2022.

[48] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 4229–4238, 2020.

[49] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Edge-assisted hierarchical federated learning with non-iid data," *arXiv preprint arXiv:1905.06641*, 2019.

[50] M. S. H. Abad, E. Ozfatura, D. GUndUz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8866–8870.

[51] W. Wen, H. H. Yang, W. Xia, and T. Q. S. Quek, "Towards fast and energy-efficient hierarchical federated edge learning: A joint design for helper scheduling and resource allocation," in *IEEE International Conference on Communications (ICC)*, 2022.

[52] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2022–2035, 2020.

[53] C. Xu, S. Liu, Z. Yang, Y. Huang, and K.-K. Wong, "Learning rate optimization for federated learning exploiting over-the-air computation," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3742–3756, 2021.

[54] S. Wang, Y. Hong, R. Wang, Q. Hao, Y.-C. Wu, and D. W. K. Ng, "Edge federated learning via unit-modulus over-the-air computation," *IEEE Transactions on Communications*, vol. 70, no. 5, pp. 3141–3156, 2022.

[55] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2022.

[56] T. Yu, E. Bagdasaryan, and V. Shmatikov, "Salvaging federated learning by local adaptation," *arXiv preprint arXiv:2002.04758*, 2020.

[57] K. Wang, R. Mathews, C. Kiddon, H. Eichner, F. Beaufays, and D. Ramage, "Federated evaluation of on-device personalization," *arXiv preprint arXiv:1910.10252*, 2019.

[58] Y. Jiang, J. Konečnỳ, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," *arXiv preprint arXiv:1909.12488*, 2019.

[59] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," *arXiv preprint arXiv:1802.07876*, 2018.

[60] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3557–3568, 2020.

[61] G. Cheng, K. Chadha, and J. Duchi, "Fine-tuning is fine in federated learning," *arXiv preprint arXiv:2108.07313*, 2021.

[62] F. Hanzely and P. Richtárik, "Federated learning of a mixture of global and local models," *arXiv preprint arXiv:2002.05516*, 2020.

[63] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," in *Proc. Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 21394-21405, 2020.

[64] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *Proc. International Conference on Machine Learning (ICML)*, 2021.

[65] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proc. Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4427-4437.

[66] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal, "Federated multi-task learning under a mixture of distributions," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 15 434–15 447.

[67] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3710–3722, 2021.

[68] H. Yang, Z. Liu, T. Quek, and H. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2020.

[69] S. Wang, T. Tuor, T. Salonidis, K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[70] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *IEEE International Conference on Communications (ICC)*, pp. 1-7, 2019.

[71] A. Reisizadeh, I. Tziotis, H. Hassani, A. Mokhtari, and R. Pedarsani, "Adaptive node participation for straggler-resilient federated learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 8762–8766.

[72] W. Shi, S. Zhou, and Z. Niu, "Device scheduling with fast convergence for wireless federated learning," in *IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[73] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *IEEE International Conference on Big Data*, 2020, pp. 15–24.

[74] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "Safa: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, 2020.

[75] M. R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, "Asynchronous federated learning for geospatial applications," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018, pp. 21–28.

[76] B. Woodworth, K. K. Patel, S. Stich, Z. Dai, B. Bullins, B. Mcmahan, O. Shamir, and N. Srebro, "Is local SGD better than minibatch SGD?" in *International Conference on Machine Learning (ICML)*, vol. 119, pp. 10 334–10 343, 2020.

[77] J. Wang and G. Joshi, "Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms," *Journal of Machine Learning Research (JMLR)*, vol. 22, no. 213, pp. 1–50, 2021.

[78] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," in *International Conference on Learning Representations (ICLR)*, 2020.

[79] T. Lin, L. Kong, S. Stich, and M. Jaggi, "Extrapolation for large-batch training in deep learning," in *International Conference on Machine Learning (ICML)*, vol. 119, 2020, pp. 6094–6104.

[80] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[81] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[82] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[83] D. Blalock, J. Ortiz, J. Frankler, and J. Guttag, "What is the state of neural network pruning?" in *Conference on Machine Learning and Systems (MLSys)*, 2020.

[84] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2022.

[85] S. Itahara, T. Nishio, M. Morikura, and K. Yamamoto, "Lottery hypothesis based unsupervised pre-training for model compression in federated learning," in *IEEE Vehicular Technology Conference (VTC)*, 2020, pp. 1–5.

[86] A. Li, J. Sun, B. Wang, L. Duan, S. Li, Y. Chen, and H. Li, "Lotteryfl: Empower edge intelligence with personalized and communication-efficient federated learning," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 68–79.

[87] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, pp. 1–13, 2014.

[88] S. Golomb, "Run-length encodings (corresp.)," *IEEE Trans. Inf. Theor.*, vol. 12, no. 3, pp. 399–401, Sep 2006.

[89] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *International Conference on Learning Representations (ICLR)*, 2018.

[90] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[91] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[92] M. Abadi, A. Agarwal, P. Barham *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[93] P. K. Mudrakarta, M. Sandler, A. Zhmoginov, and A. Howard, "K for the price of 1: Parameter efficient multi-task and transfer learning," in *Proc. International Conference on Learning Representations (ICLR)*, 2019.

[94] T. Li, A. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

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

[96] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2015, pp. 448–456.

[97] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, 2021.

[98] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 1, pp. 212–229, 2019.

[99] S. Caldas, P. Wu, T. Li, J. Konecný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," in *NeurIPS Workshop on Federated Learnign for Data Privacy and Confidentiality*, 2019.

[100] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, no. 5, pp. 1–17, 1964.

[101] T. Tieleman and G. Hinton, "Rmsprop, coursera: Neural netwroks for machine learning," *Technical Report*, 2012.

[102] W. Chen, K. Bhardwaj, and R. Marculescu, "Fedmax: mitigating activation divergence for accurate and communication-efficient federated learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2020, pp. 348–363.

[103] A. Défossez, L. Bottou, F. Bach, and N. Usunier, "A simple convergence proof of adam and adagrad," *arXiv e-prints arXiv:2003.02395*, 2020.

[104] R. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley - benchmarking deep learning optimizers," in *Proc. International Conference on Machine Learning (ICML)*, vol. 139, 2021, pp. 9367–9376.

[105] D. Zhou, J. Chen, Y. Cao, Y. Tang, Z. Yang, and Q. Gu, "On the convergence of adaptive gradient methods for nonconvex optimization," in *NeurIPS Workshop on Optimization for Machine Learning (OPT)*, 2020.

[106] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning," in *Proceedings of Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 7232–7241.

[107] M. Song, Z. Wang, Z. Zhang, Y. Song, Q. Wang, J. Ren, and H. Qi, "Analyzing user-level privacy attack against federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2430–2444, 2020.

[108] D. Hernandez and T. B. Brown, "Measuring the algorithmic efficiency of neural networks," *arXiv e-prints arXiv:2005.04305*, 2020.

Appendix A

Chapter 5 Supplementary Material

A.1 Key Lemmas

Previously, Li *et al.* [28] analysed the per-iteration convergence of FedAvg for μ-strongly convex functions when using a decreasing stepsize. Their result was the first to prove convergence for non-IID clients with partial participation. The assumptions in Chapter 5 are at least as strong as Li *et al.*, so their intermediary result bounding the distance to the global minimiser when using partial client participation can be directly used.

Lemma A.1: *Given Assumptions 1-4, the expected distance between average client model $\bar{\boldsymbol{x}}_t$ and the global minimiser $\boldsymbol{x}^*$ is upper-bounded by:*

$$\mathbb{E}\left[\|\bar{\boldsymbol{x}}_{t+1} - \boldsymbol{x}^*\|^2\right] \leq (1 - \eta_t\mu)\mathbb{E}\left[\|\bar{\boldsymbol{x}}_t - \boldsymbol{x}^2\|^2\right] + \eta_t^2\Big(\sum_{c=1}^{C} p_c^2\sigma_c^2 + 6L\Gamma + 8(K_t - 1)^2G^2 + \frac{C-N}{N-1}\frac{4}{N}K_t^2G^2\Big). \quad \text{(A.1)}$$

Proof: See Appendix B.3 of [28].

Lemma A.2: *Given Assumptions 1-4, the sum of expected gradient norms over T iterations of the average client model $\bar{\boldsymbol{x}}_t$ is upper-bounded by:*

$$\sum_{t=1}^{T}\eta_t\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right] \leq 2\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*) + \kappa L\Big(\sum_{c=1}^{C} p_c^2\sigma_c^2 + 6L\Gamma + 8(K_t - 1)^2G^2 + \frac{4}{N}K_t^2G^2\Big)\sum_{t=1}^{T}\eta_t^2. \quad \text{(A.2)}$$

Proof : Rearranging Lemma A.1 and then defining for notational convenience:

$$D = \left(\sum_{c=1}^{C} p_c^2\sigma_c^2 + 6L\Gamma + 8(K_t - 2)^2G^2 + \frac{C-N}{N-1}\frac{4}{N}K_t^2G^2\right),$$

the recursive definition can be written as:

$$\eta_t\mu\mathbb{E}\left[\|\bar{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2\right] \leq \mathbb{E}\left[\|\bar{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2\right] - \mathbb{E}\left[\|\bar{\boldsymbol{x}}_{t+1} - \boldsymbol{x}^*\|^2\right] + \eta_t^2 D. \quad \text{(A.3)}$$

Using Assumption 1 (L-smoothness), then:

$$\frac{\eta_t\mu}{L^2}\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right] \leq \mathbb{E}\left[\|\bar{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2\right] - \mathbb{E}\left[\|\bar{\boldsymbol{x}}_{t+1} - \boldsymbol{x}^*\|^2\right] + \eta_t^2 D. \tag{A.4}$$

Summing up the T iterations and telescoping the distance terms gives:

$$\frac{\mu}{L^2}\sum_{t=1}^{T}\eta_t^2\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right] \leq \mathbb{E}\left[\|\bar{\boldsymbol{x}}_0 - \boldsymbol{x}^*\|^2\right] - \mathbb{E}\left[\|\bar{\boldsymbol{x}}_T - \boldsymbol{x}^*\|^2\right] + D\sum_{t=1}^{T}\eta_t^2. \tag{A.5}$$

Using Assumption 1 (L-smoothness) and Assumption 2 (μ-strong convexity) to bound the distance terms now gives:

$$\frac{\mu}{L^2}\sum_{t=1}^{T}\eta_t^2\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right] \leq \frac{2}{\mu}\left[F(\bar{\boldsymbol{x}}_0) - F^*\right] - \frac{2}{L}\left[F(\bar{\boldsymbol{x}}_T) - F^*\right] + D\sum_{t=1}^{T}\eta_t^2, \tag{A.6}$$

which can be simplified by noting that $\mu \leq L$, so that:

$$\frac{\mu}{L^2}\sum_{t=1}^{T}\eta_t^2\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right] \leq \frac{2}{\mu}F(\bar{\boldsymbol{x}}_0) - \frac{2}{L}F(\bar{\boldsymbol{x}}_T) + D\sum_{t=1}^{T}\eta_t^2. \tag{A.7}$$

Multiplying both sides of the inequality by L^2/μ, using the lower-bound $F^* \leq F(\bar{\boldsymbol{x}}_T)$, the definition $\kappa = L/\mu$, and the fact that $\frac{C-N}{N-1} \leq 1$ completes the proof.

A.2 Proof of Theorem 5.1

The bound on gradient norms given in Lemma A.2 uses the index t that denotes the global SGD step that each client evaluates (irrespective of communication round). However, FedAvg clients participate in communication rounds. The values of η_t and K_t are therefore fixed within each communication round. To account for this, Lemma A.2 can be reindexed using the given communication round r and local step k: $t = I + k$, where $I = \sum_{i=1}^{r} K_i$. The total number of communication rounds is R, therefore $T = \sum_{r=1}^{R} K_r$. Using this to reindex Lemma A.2:

$$\begin{aligned}\sum_{r=1}^{R}\eta_r\sum_{k=1}^{K_r}\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_{I+k})\|^2\right] &\leq 2\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*) \\ &\quad + \kappa L\Big(\sum_{c=1}^{C}p_c^2\sigma_c^2 + 6L\Gamma + 8(K_r-1)^2G^2 + \frac{4}{N}K_r^2G^2\Big)\sum_{r=1}^{R}\eta_r^2K_r \\ &\leq 2\kappa\left(F(\bar{\boldsymbol{x}}_0) - F^*\right) + \kappa L\left(\sum_{n=1}^{N}p_n^2\sigma_n^2 + 6L\Gamma\right)\sum_{r=1}^{R}\eta_r^2K_r + 16\kappa LG^2\sum_{r=1}^{R}\eta_r^2K_r^3.\end{aligned} \tag{A.8}$$

Diving both sides through by $\sum_{r=1}\eta_r K_r$:

$$\begin{aligned}\frac{\sum_{r=1}^{R}\eta_r\sum_{k=1}^{K_r}\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_{I+k})\|^2\right]}{\sum_{r=1}^{R}\eta_rK_r} &\leq \frac{2\kappa\left(F(\bar{\boldsymbol{x}}_0) - F^*\right)}{\sum_{r=1}^{R}\eta_rK_r} \\ &\quad + \kappa L\left(\sum_{n=1}^{N}p_n^2\sigma_n^2 + 6L\Gamma\right)\frac{\sum_{r=1}^{R}\eta_r^2K_r}{\sum_{r=1}^{R}\eta_rK_r} + 16\kappa LG^2\frac{\sum_{r=1}^{R}\eta_r^2K_r^3}{\sum_{r=1}^{R}\eta_rK_r}.\end{aligned} \tag{A.9}$$

Using a fixed $\eta_r = \eta \leq 1/4L$, then the above inequality can be simplified as:

$$\frac{\sum_{r=1}^{R}\sum_{k=1}^{K_r}\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_{I+k})\|^2\right]}{\sum_{r=1}^{R}K_r} \leq \frac{2\kappa\left(F(\bar{\boldsymbol{x}}_0)-F^*\right)}{\eta\sum_{r=1}^{R}K_r} + \eta\kappa L\left(\sum_{n=1}^{N}p_n^2\sigma_n^2 + 6L\Gamma\right) + 16\eta\kappa LG^2\frac{\sum_{r=1}^{R}K_r^3}{\sum_{r=1}^{R}K_r}. \tag{A.10}$$

Reindexing using the fact that $\sum_{r=1}^{R}K_r = T$ gives:

$$\frac{\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]}{T} \leq \frac{2\kappa\left(F(\bar{\boldsymbol{x}}_0)-F^*\right)}{\eta T} + \eta\kappa L\left[\sum_{n=1}^{N}p_n^2\sigma_n^2 + 6L\Gamma + 16G^2\frac{\sum_{r=1}^{R}K_r^3}{\sum_{r=1}^{R}K_r}\right]. \tag{A.11}$$

Using $\min\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\} \leq \mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]$ then completes the proof.

A.3 Proof of Theorem 5.2

Starting from the bound on gradient norms using a constant K and η (8):

$$\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\} \leq \frac{2\kappa(\kappa F(\bar{\boldsymbol{x}}_0)-F^*)}{\eta WK}\left[\frac{|\boldsymbol{x}|}{D}+\frac{|\boldsymbol{x}|}{U}+\beta K\right] + \eta\kappa L\left[\sum_{c=1}^{C}p_c^2\sigma_c^2 + 6L\Gamma + \left(8+\frac{4}{N}\right)G^2K^2\right]. \tag{A.12}$$

Taking the first derivative with respect to K gives:

$$\frac{d\,\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}}{dK} = \frac{-2\kappa(\kappa F(\bar{\boldsymbol{x}}_0)-F^*)}{\eta WK^2}\left[\frac{|\boldsymbol{x}|}{D}+\frac{|\boldsymbol{x}|}{U}+\beta K\right] + 2\eta\kappa L\left(8+\frac{4}{N}\right)G^2K. \tag{A.13}$$

Taking the second derivative with respect to K gives:

$$\frac{d^2\,\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}}{dK^2} = \frac{4\kappa(\kappa F(\bar{\boldsymbol{x}}_0)-F^*)}{\eta WK^3}\left[\frac{|\boldsymbol{x}|}{D}+\frac{|\boldsymbol{x}|}{U}+\beta K\right] + 2\eta\kappa L\left(8+\frac{4}{N}\right)G^2. \tag{A.14}$$

Considering $(F(\bar{\boldsymbol{x}}_0)-F^*) > 0$ and all the constants in (25) are > 0, then inspection of (25) shows that the second derivative with respect to K is greater than 0, and hence (23) is convex. Solving $d\,\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}/dK = 0$ gives Theorem 2.

A.4 Proof of Corollary 5.2.1

As with the proof of Theorem 2, start with the bound on gradient norms using a constant K and η given in (8):

$$\min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\} \leq \frac{2\kappa(\kappa F(\bar{\boldsymbol{x}}_0)-F^*)}{\eta WK}\left[\frac{|\boldsymbol{x}|}{D}+\frac{|\boldsymbol{x}|}{U}+\beta K\right] + \eta\kappa L\left[\sum_{c=1}^{C}p_c^2\sigma_c^2 + 6L\Gamma + \left(8+\frac{4}{N}\right)G^2K^2\right]. \tag{A.15}$$

Taking the first derivative with respect to η gives:

$$\frac{d \min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}}{d\,\eta} = \frac{-2\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{\eta^2 WK}\left[\frac{|\boldsymbol{x}|}{D} + \frac{|\boldsymbol{x}|}{U} + \beta K\right] + \kappa L\left[\sum_{c=1}^{C} p_c^2\sigma_c^2 + 6L\Gamma + \left(8 + \frac{4}{N}\right)G^2K^2\right]. \quad \text{(A.16)}$$

Taking the second derivative with respect to η gives:

$$\frac{d^2 \min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}}{d\,\eta^2} = \frac{4\kappa(\kappa F(\bar{\boldsymbol{x}}_0) - F^*)}{\eta^3 WK}\left[\frac{|\boldsymbol{x}|}{D} + \frac{|\boldsymbol{x}|}{U} + \beta K\right]. \quad \text{(A.17)}$$

Noting that $(F(\bar{\boldsymbol{x}}_0) - F^*) > 0$ and all the constants in (28) are > 0, then inspection of (28) shows that the second derivative with respect to η is > 0 and hence (27) is convex. Solving $d \min_t\{\mathbb{E}\left[\|\nabla F(\bar{\boldsymbol{x}}_t)\|^2\right]\}/d\eta = 0$ yields Corollary 2.1.

Appendix B

Chapter 6 Supplementary Material

B.1 Proof of Theorems

In the proofs below, unless otherwise specified, the expectation $\mathbb{E}[\cdot]$ is over both the clients selected randomly in each round of FedGBO, and the random sampling of minibatches in local updates.

B.1.1 Key Lemmas

Lemma B.1 (Bounding the momentum term). *Given $\tilde{\eta} = \eta K > 0$, $0 \leq \beta < 1$, $\boldsymbol{m}_t$ defined in (6.4) and (6.5), and Assumptions A1-A5, then*

$$\mathbb{E}\left[\|\boldsymbol{m}_t\|^2\right] \leq 2\frac{R^2 + \tilde{\eta}^2 Y}{(1-\beta)^2},$$

where $Y = 18B^2L^2\left(R^2 + G^2 + \frac{\sigma^2}{K}\right)$.

Proof. Taking any iteration t:

$$\begin{aligned}
\mathbb{E}\left[\|\boldsymbol{m}_t\|^2\right] &= \mathbb{E}\left[\left\|\sum_{i=0}^{t-1}\beta^i(\boldsymbol{g}_{t-i} + \boldsymbol{e}_{t-i})\right\|^2\right] \\
&\leq \left(\sum_{i=0}^{t-1}\beta^i\right)\sum_{i=0}^{t-1}\beta^i\mathbb{E}\left[\|\boldsymbol{g}_{t-i} + \boldsymbol{e}_{t-i}\|^2\right] \\
&\leq \frac{1}{1-\beta}\sum_{i=0}^{t-1}\beta^i\mathbb{E}\left[\|\boldsymbol{g}_{t-i} + \boldsymbol{e}_{t-i}\|^2\right] \\
&\leq \frac{2}{1-\beta}\sum_{i=0}^{t-1}\beta^i\left(\mathbb{E}\left[\|\boldsymbol{g}_{t-i}\|^2\right] + \mathbb{E}\left[\|\boldsymbol{e}_{t-i}\|^2\right]\right) \\
&\leq 2\frac{\mathbb{E}\left[\|\boldsymbol{g}_{t-i}\|^2\right] + \mathbb{E}\left[\|\boldsymbol{e}_{t-i}\|^2\right]}{(1-\beta)^2} \\
&\leq 2\frac{R^2 + \tilde{\eta}^2 Y}{(1-\beta)^2}.
\end{aligned}$$

□

Here, the first inequality comes from Jensen, the third inequality is from Cauchy-Schwarz and the linearity of expectation, and the last inequality from Assumption 3 and the definition of Y.

Lemma B.2 (bounding the negative term of the Descent Lemma). *Given $\tilde{\eta} = \eta K > 0$, $0 \leq \beta < 1$, $\boldsymbol{m}_t$ defined in (6.4) and (6.5), and Assumptions A1-A5, then*

$$\mathbb{E}[\nabla F(\boldsymbol{x}_{t-1})^\top \boldsymbol{m}_t] \geq \frac{1}{2}\sum_{k=0}^{t-1} \beta^k \mathbb{E}\left[\|\nabla F(\boldsymbol{x}_{t-k-1})\|^2\right] - \frac{\tilde{\eta}^2 Y}{2(1-\beta)} - \frac{2\tilde{\eta}L\beta(R^2+\tilde{\eta}^2 Y)}{(1-\beta)^3}.$$

where $Y = 18B^2L^2\left(R^2 + G^2 + \frac{\sigma^2}{K}\right)$.

Proof. To ease notation, denote $\boldsymbol{G}_t = \nabla F(\boldsymbol{x}_{t-1})$ and $\boldsymbol{c}_t = \boldsymbol{g}_t + \boldsymbol{e}_t$:

$$\begin{aligned}\boldsymbol{G}_t^\top \boldsymbol{m}_t &= \sum_{k=0}^{t-1} \beta^k \boldsymbol{G}_t^\top \boldsymbol{c}_t \\ &= \sum_{k=0}^{t-1} \beta^k \boldsymbol{G}_{t-k}^\top \boldsymbol{c}_t + \sum_{k=0}^{t-1} \beta^k (\boldsymbol{G}_t - \boldsymbol{G}_{t-k})^\top \boldsymbol{c}_t. \end{aligned} \tag{B.1}$$

Due to F being L-smooth (Assumption 4), and using the relaxed triangle inequality:

$$\begin{aligned}\|\boldsymbol{G}_t - \boldsymbol{G}_{t-k}\|^2 &\leq L^2 \left\|\sum_{l=1}^{k} \tilde{\eta}\boldsymbol{m}_{n-l}\right\|^2 \\ &\leq \tilde{\eta}^2 L^2 k \sum_{l=1}^{k} \|\boldsymbol{m}_{n-l}\|^2. \end{aligned} \tag{B.2}$$

Using $(\lambda\boldsymbol{x} - \boldsymbol{y})^2 \geq 0, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d, \lambda > 0$, and hence $\|\boldsymbol{x}\boldsymbol{y}\| \leq \frac{\lambda}{2}\|\boldsymbol{x}\|^2 + \frac{1}{2\lambda}\|\boldsymbol{y}\|^2$, then $\boldsymbol{x} = G_t - G_{t-k}$, $\boldsymbol{y} = \boldsymbol{c}_t$, $\lambda = \frac{1-\beta}{k\tilde{\eta}L}$, can be substituted into (B.3):

$$\boldsymbol{G}_t^\top \boldsymbol{m}_t \geq \sum_{k=0}^{t-1} \beta^k \boldsymbol{G}_{t-k}^\top \boldsymbol{c}_{t-k} - \sum_{k=1}^{t-1} \frac{\beta^k}{2}\left(\left((1-\beta)\tilde{\eta}L\sum_{l=1}^{k}\|\boldsymbol{m}_{t-l}\|^2\right) + \frac{\tilde{\eta}Lk}{1-\beta}\|\boldsymbol{c}_{t-k}\|^2\right). \tag{B.3}$$

Taking the expectation of both sides gives:

$$\begin{aligned}\mathbb{E}[\boldsymbol{G}_t^\top \boldsymbol{m}_t] \geq \sum_{k=0}^{t-1} \beta^k \mathbb{E}[\boldsymbol{G}_{t-k}^\top \boldsymbol{c}_{t-k}] - \tilde{\eta}L\sum_{k=1}^{t-1}\frac{\beta^k}{2}\Bigg(\Bigg((1-\beta)\sum_{l=1}^{k}\mathbb{E}[\|\boldsymbol{m}_{t-l}\|^2]\Bigg) \\ + \frac{k}{1-\beta}\mathbb{E}[\|\boldsymbol{c}_{t-k}\|^2]\Bigg). \end{aligned} \tag{B.4}$$

Bounding the $\mathbb{E}[\boldsymbol{G}_{t-k}^\top \boldsymbol{c}_{t-k}]$ term, this time using $(\boldsymbol{G}_{t-k}^\top + \boldsymbol{e}_{t-k})^2 \geq 0$, and hence $\boldsymbol{G}_{t-k}^\top \boldsymbol{e}_{t-k} \geq -\frac{1}{2}(\|\boldsymbol{G}_{t-k}\|^2 + \|\boldsymbol{e}_{t-k}\|^2)$, and linearity of expectation:

$$\begin{aligned}\mathbb{E}[\boldsymbol{G}_{t-k}^\top \boldsymbol{c}_{t-k}] &= \mathbb{E}\left[\boldsymbol{G}_{t-k}^\top(\boldsymbol{g}_{t-k} + \boldsymbol{e}_{t-k})\right] \\ &= \mathbb{E}\left[\boldsymbol{G}_{t-k}^\top \boldsymbol{g}_{t-k}\right] + \mathbb{E}\left[\boldsymbol{G}_{t-k}^\top \boldsymbol{e}_{t-k}\right] \\ &\geq \mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2\right] - \frac{1}{2}\mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2 + \|\boldsymbol{e}_{t-k}\|^2\right] \\ &= \frac{1}{2}\left(\mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2\right] - \mathbb{E}\left[\|\boldsymbol{e}_{t-k}\|^2\right]\right). \end{aligned} \tag{B.5}$$

Using Cauchy-Schwarz, linearity of expectation, (6.3), Assumption 3, and the definition of Y:

$$\begin{aligned}\mathbb{E}[\|\boldsymbol{c}_{t-k}\|^2] &= \mathbb{E}[\|\boldsymbol{g}_{t-k} + \boldsymbol{e}_{t-k}\|^2] \\ &\leq 2(\mathbb{E}[\|\boldsymbol{g}_{t-k}\|^2] + \mathbb{E}[\|\boldsymbol{e}_{t-k}\|^2]) \\ &\leq 2(R^2 + \tilde{\eta}^2 Y). \end{aligned} \tag{B.6}$$

Then inserting Lemma B.1, (B.5) and (B.6) into (B.4) gives:

$$\begin{aligned}\mathbb{E}[\boldsymbol{G}_t^\top \boldsymbol{m}_t] &\geq \sum_{k=0}^{t} \frac{\beta^k}{2} \left(\mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] - \mathbb{E}[\|\boldsymbol{e}_{t-k}\|^2]\right) \\ &\quad - \tilde{\eta} L \sum_{k=1}^{t-1} \frac{\beta^k}{2} \left(\left((1-\beta) \sum_{l=1}^{k} 2 \frac{R^2 + \tilde{\eta}^2 Y}{(1-\beta)^2} \right) + \frac{k}{1-\beta} 2(R^2 + \tilde{\eta}^2 Y) \right) \\ &= \sum_{k=0}^{t} \frac{\beta^k}{2} \mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] - \sum_{k=0}^{t} \frac{\beta^k}{2} \mathbb{E}[\|\boldsymbol{e}_{t-k}\|^2] \\ &\quad - \tilde{\eta} L (R^2 + \tilde{\eta}^2 Y) \sum_{k=1}^{t-1} \beta^k \left(\left(\sum_{l=1}^{k} \frac{1}{(1-\beta)} \right) + \frac{k}{1-\beta} \right) \\ &= \sum_{k=0}^{t} \frac{\beta^k}{2} \mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] - \frac{\tilde{\eta}^2 Y}{2(1-\beta)} - \frac{2\tilde{\eta} L}{1-\beta} (R^2 + \tilde{\eta}^2 Y) \sum_{k=1}^{t-1} \beta^k k. \end{aligned} \tag{B.7}$$

Using Lemma B.2 from [2], which states that, for $0 < a < 1, i \in \mathbb{N}, Q \geq i$:

$$\sum_{q=i}^{Q} a^q q \leq \frac{a}{(1-a)^2}, \tag{B.8}$$

then (B.7) becomes:

$$\mathbb{E}[\boldsymbol{G}_t^\top \boldsymbol{m}_t] \geq \frac{1}{2} \sum_{k=0}^{t-1} \beta^k \mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] - \frac{\tilde{\eta}^2 Y}{2(1-\beta)} - \frac{2\tilde{\eta} L \beta (R^2 + \tilde{\eta}^2 Y)}{(1-\beta)^3}. \tag{B.9}$$

Substituting the definition of $\boldsymbol{G}_t$ completes the Lemma. □

B.1.2 Proof of Theorem 6.1

Theorem 6.1 (Convergence of FedGBO using SGDm) *Let the assumptions A1-A5 hold, the total number of communication rounds $T > (1-\beta)^{-1}$, $\tilde{\eta} = K\eta > 0, 0 \leq \beta < 1$, and using the update steps given by (6.4) and (6.5), then:*

$$\mathbb{E}[\|\nabla F(\boldsymbol{x}_t)\|^2] \leq \frac{2(1-\beta)}{\tilde{\eta}\tilde{T}} (F(\boldsymbol{x}_0) - F^*) + \frac{T\tilde{\eta}^2 Y}{\tilde{T}} + \frac{2T\tilde{\eta} L (1+\beta)(R^2 + \tilde{\eta}^2 Y)}{\tilde{T}(1-\beta)},$$

where $\tilde{T} = T - \frac{\beta}{1-\beta}$, and $Y = 18B^2L^2 \left(R^2 + G^2 + \frac{\sigma^2}{K}\right)$.

Proof. Using the L-smoothness of F, and the update rules from (6.4) and (6.5), the Descent Lemma of FedGBO with SGDm is given by:

$$F(\boldsymbol{x}_t) \leq F(\boldsymbol{x}_{t-1}) - \tilde{\eta} \boldsymbol{G}_t^\top \boldsymbol{m}_t + \frac{\tilde{\eta}^2 L}{2} \|\boldsymbol{m}_t\|^2. \tag{B.10}$$

Taking expectations of both sides, and inserting Lemma B.1 and Lemma B.2 into (B.10):

$$\begin{aligned}\mathbb{E}[F(\boldsymbol{x}_t)] &\leq \mathbb{E}[F(\boldsymbol{x}_{t-1})] - \frac{\tilde{\eta}}{2}\sum_{k=0}^{t-1}\beta^k\mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] \\ &\quad + \frac{\tilde{\eta}^3 Y}{2(1-\beta)} + \frac{2\tilde{\eta}^2 L\beta(R^2+\tilde{\eta}^2 Y)}{(1-\beta)^3} + \frac{\tilde{\eta}^2 L(R^2+\tilde{\eta}^2 Y)}{(1-\beta)^2} \\ &= \mathbb{E}[F(\boldsymbol{x}_{t-1})] - \frac{\tilde{\eta}}{2}\sum_{k=0}^{t-1}\beta^k\mathbb{E}[\|\boldsymbol{G}_{t-k}\|^2] + \frac{\tilde{\eta}^3 Y}{2(1-\beta)} \\ &\quad + \frac{\tilde{\eta}^2 L(1+\beta)(R^2+\tilde{\eta}^2 Y)}{(1-\beta)^3}. \end{aligned} \tag{B.11}$$

Rearranging (B.11), summing over the iterations $t = 1, \cdots, T$ and telescoping:

$$\begin{aligned}\frac{\tilde{\eta}}{2}\sum_{t=1}^{T}\sum_{k=0}^{t-1}\beta^k\mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2\right] &\leq F(\boldsymbol{x}_0) - \mathbb{E}\left[F(\boldsymbol{x}_T)\right] + \frac{\tilde{\eta}^3 TY}{2(1-\beta)} \\ &\quad + \frac{\tilde{\eta}^2 LT(1+\beta)(R^2+\tilde{\eta}^2 Y)}{(1-\beta)^3}. \end{aligned} \tag{B.12}$$

Proceeding as [2] do to bound the left hand side of (B.12), introduce the change of index $i = t - k$:

$$\begin{aligned}\frac{\tilde{\eta}}{2}\sum_{t=1}^{T}\sum_{k=0}^{t-1}\beta^k\mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2\right] &= \frac{\tilde{\eta}}{2}\sum_{t=1}^{T}\sum_{i=0}^{t}\beta^{t-i}\mathbb{E}\left[\|\boldsymbol{G}_i\|^2\right] \\ &= \frac{\tilde{\eta}}{2}\sum_{i=0}^{T}\mathbb{E}\left[\|\boldsymbol{G}_i\|^2\right]\sum_{t=1}^{T}\beta^{t-i} \\ &= \frac{\tilde{\eta}}{2(1-\beta)}\sum_{i=1}^{T}\left(\mathbb{E}\left[\|\nabla F(\boldsymbol{x}_{i-1})\|^2\right](1-\beta^{T-i+1})\right) \\ &= \frac{\tilde{\eta}}{2(1-\beta)}\sum_{i=0}^{T-1}\left(\mathbb{E}\left[\|\nabla F(\boldsymbol{x}_i)\|^2\right](1-\beta^{T-i})\right). \end{aligned} \tag{B.13}$$

(B.13) shows a non-normalised iteration probability as defined in Section 6.2.2 ($\mathbb{P}[t = j] \propto 1 - \beta^{T-j}$). The normalisation constant for summing to 1 is:

$$\sum_{i=0}^{T-1} 1 - \beta^{T-i} = T - \beta\frac{1-\beta^T}{1-\beta} \geq T - \frac{\beta}{1-\beta} = \tilde{T}. \tag{B.14}$$

(B.14) can then be used in (B.13) to obtain:

$$\frac{\tilde{\eta}}{2}\sum_{t=1}^{T}\sum_{k=0}^{t-1}\beta^k\mathbb{E}\left[\|\boldsymbol{G}_{t-k}\|^2\right] \geq \frac{\tilde{\eta}\tilde{T}}{2(1-\beta)}\mathbb{E}\left[\|\nabla F(\boldsymbol{x}_t)\|^2\right]. \tag{B.15}$$

Inserting (B.15) into (B.13), and using the lower bound $F(\boldsymbol{x}) \geq F^*$ completes the proof:

$$\mathbb{E}[\|\nabla F(\boldsymbol{x}_t)\|^2] \leq \frac{2(1-\beta)}{\tilde{\eta}\tilde{T}}(F(\boldsymbol{x}_0) - F^*) + \frac{T\tilde{\eta}^2 Y}{\tilde{T}} + \frac{2T\tilde{\eta}L(1+\beta)(R^2+\tilde{\eta}^2 Y)}{\tilde{T}(1-\beta)^2}. \quad \square$$

B.2 Computation Costs of FL Algorithms

The total number of Floating Point Operations (FLOPs) required on average to compute the local update of the FL algorithms studied in Chapter 6 were determined. Tables B.2 and B.3 shows these values for $K = 10$ and $K = 50$ local steps, respectively. The total FLOPs were calculated as the sum of FLOPs required to compute K minibatch gradients [108] and update the local model using the algorithm's optimisation strategy. The Mimelite algorithm also requires computing full-batch gradients. The cost of work on the server as it is represents a small fraction of the total, and in FL the server is assumed to have arbitrarily high compute power compared to client devices. FLOPs are calculated from an algorithmic standpoint, and do not consider optimisations such as vectorised operations, memory access etc.

The total FLOPs for a single forward-backward pass were computed using the Thop library (`https://github.com/Lyken17/pytorch-OpCounter/`), and the formula from the following OpenAI page: `https://openai.com/blog/ai-and-compute/`.

Local update steps involve computing gradients and applying them to the local model, plus any updates to a local optimiser if used. For example, for FedGBO using RMSProp, Table 6.1 shows that there are 5 operations per step that act on either $\boldsymbol{g}$ or $\boldsymbol{v}$: $(+)$, $(\times -\eta)$, $(\div)$, $(\sqrt{\boldsymbol{v}})$, $(+\epsilon)$. MFL with RMSProp has 8 total operations: those 5 plus 3 to track the local $\boldsymbol{v}$ values. FedGBO, Mime and MFL all have 4 operations per SGDm update, as MFL's local tracking can be incorporated into the model update. FedAvg, AdaptiveFedOpt and FedMAX use vanilla SGD with 2 operations: $(+)$, $(\times -\eta)$. FedProx uses 5 operates due to the proximal update. Finally, Mimelite adds the cost of computing full-batch gradients.

The formulas for computing the FLOPs for the local update of each algorithm, N, are therefore as follows:

$$\begin{aligned}
\mathrm{N}_{\mathrm{FedGBO}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + C_{\mathrm{Fixed}}|\boldsymbol{x}|),\\
\mathrm{N}_{\mathrm{Mimelite}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + C_{\mathrm{Fixed}}|\boldsymbol{x}|)\\
&\quad + n(\mathrm{fwd}+\mathrm{bwd}),\\
\mathrm{N}_{\mathrm{MFL}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + C_{\mathrm{Moving}}|\boldsymbol{x}|),\\
\mathrm{N}_{\mathrm{AdaptiveFedOpt}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + 2|\boldsymbol{x}|),\\
\mathrm{N}_{\mathrm{SGD}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + 2|\boldsymbol{x}|),\\
\mathrm{N}_{\mathrm{FedProx}} &= K(B(\mathrm{fwd}+\mathrm{bwd}) + 5|\boldsymbol{x}|),
\end{aligned} \tag{B.16}$$

where K is the number of local updates, B is the minibatch size, $(\mathrm{fwd}+\mathrm{bwd})$ are the FLOPs required to compute the forward plus backward passes on one sample, C_* is the number of operations for the type of local adaptive optimisation (see Table B.1), $|\boldsymbol{x}|$ is the number of model parameters, and n is the average number of client samples, as shown in Table 6.3.

Table B.1: Number of operations per local step of fixed or moving local optimisers.

Algorithm	C_{Fixed}	C_{Moving}
RMSProp	5	5
SGDm	5	8
Adam	8	11

Table B.2: Total average FLOPs ($\times 10^8$) performed per client per round for different FL tasks, using different optimisers and FL algorithms: FedGBO, Mimelite, MFL using RMSProp, SGDm and Adam. AdaptiveFedOpt, FedAvg and FedMAX use SGD and FedProx. Values shown for $K = 10$ local update steps, to 3 significant figures.

Task	RMSProp	SGDm	Adam	SGD	FedProx
CIFAR100	50.2	50.2	50.6	49.8	50.2
	65.7	65.7	66.1		
	50.2	50.6	50.9		
Sent140	0.0290	0.0290	0.0320	0.0260	0.0290
	0.0350	0.0350	0.0365		
	0.0290	0.0320	0.0350		
FEMNIST	32.2	32.2	32.4	31.9	32.2
	49.0	49.0	49.3		
	32.2	32.4	32.7		
Shakespeare	118	118	118	118	118
	2180	2180	2180		
	118	118	118		

Table B.3: Total average FLOPs ($\times 10^8$) performed per client per round for different FL tasks, using different optimisers and FL algorithms: FedGBO, Mimelite, MFL using RMSProp, SGDm and Adam. AdaptiveFedOpt and FedAvg use SGD. Values shown for $K = 50$ local update steps, to 3 significant figures.

Task	RMSProp	SGDm	Adam	SGD	FedProx
CIFAR100	251	251	253	249	251
	266	266	268		
	251	253	255		
Sent140	0.145	0.145	0.160	0.130	0.145
	0.150	0.150	0.165		
	0.145	0.160	0.175		
FEMNIST	161	161	162	160	161
	178	178	179		
	161	162	163		
Shakespeare	592	592	592	591	592
	2650	2650	2650		
	592	592	592		