Elastic Optimization for Stragglers in Edge Federated Learning

A thesis presented for the fulfilment of the requirements for the degree of
Masters of Science (Research)

By

Khadija Sultana

Institute of Sustainable Industries and Liveable Cities (ISILC)

Victoria University
Melbourne, Victoria, Australia
August 2022

Abstract

With the increasing proliferation of intelligent edge devices and their applications, machine learning has contributed significantly to regression analysis. Over the past few decades, the sub-fields of machine learning have evolved, such as evolutionary computing, computer vision, natural language processing, neural net- works and speech recognition. Traditionally, machine learning is performed by collecting a huge amount of data at a centralized location, often raising privacy concerns. More recently, edge computing has been introduced where computation moves closer to the edge devices which decreases the latency associated with the centralized cloud. Moreover, edge servers help in computation offloading which decreases the data transit time to the central cloud. However, data privacy issues re- main unresolved. Hence, federated learning has emerged as a thought-provoking technology for keeping data at the source and obtaining a collaborative predictive model. Further, federated learning follows rigid client-server architecture where one server communicates with many clients. A single global server is a single point of failure. In addition, clients have to communicate with global servers and communication latency is high in this context. Therefore, to deal with these issues, federated learning is seen as a critical application in edge computing. Hence, to fully exploit the enormous data generated by devices in edge computing, edge federated learning is a promising solution. The distributed collaborative training in EFL deals with delay and privacy issues compared to traditional model training methods. However, the existence of straggling devices degrades model performance. Stragglers are caused by data and system heterogeneity. The straggler effect can be alleviated by reinforced device selection by edge servers which can solve device heterogeneity to some extent. But, the challenge of statistical heterogeneity remains unsolved.

We investigate heterogeneity in data from two aspects: high-dimensional data generated at edge devices where the number of features is greater than the number of observations and the heterogeneity caused by partial device participation. With a large number of features, the computation overhead on the devices increases, causing edge devices to become stragglers. Also, the submission of partial results causes gradients to be diverged as more local training is performed to reach local optima. In this thesis, we introduce elastic optimization for stragglers due to data heterogeneity in edge federated learning. Specifically, we define the problem of stragglers in edge federated learning. Then, we formulate an optimization problem to be solved at the edge devices. We customize the benchmark algorithm, FedAvg, to obtain a new elastic optimization algorithm (FedEN) which is applied in the local training of edge devices. FedEN eradicates stragglers by achieving a balance between lasso and ridge penalization, thereby generating sparse model updates and forcing the parameters to be as close as possible to local optima. We experiment on the MNIST and CIFAR-10 datasets for the proposed model. Simulated experiments demonstrate that the proposed approach improves the run time training performance by achieving target accuracy in less communication rounds. The results confirm the improved performance of the FedEN approach over benchmark algorithms.

Student Declaration

"I, Khadija Sultana , declare that the Master by Research thesis entitled 'Elastic Optimization for Stragglers in Edge Federated Learning' is no more than 60,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work".

I have conducted my research in alignment with the Australian Code for the Responsible Conduct of Research and Victoria University's Higher Degree by Research Policy and Procedures.

Signature: KHADIJA SULTANA Date: **24-Aug-2022**

[redacted]

Acknowledgements

I would like to express my sincerest and deepest gratitude and appreciation to Prof. Hua Wang, Dr Khandakar Ahmed and Dr Bruce Gu. They have been a wonderful source of help and support to me, both in research and continual guidance. They taught me how to identify practical challenges and formulate them in abstract senses. While we may not be able to solve the whole abstract problem, they believe that formulating a well-defined theoretical problem and solving it partially may offer even more value than solving an extremely hard abstract problem that has little practical impact. I was also inspired by their ability to identify the student's interest and I deeply appreciate them allowing me to explore different research areas and establish fruitful collaborations. Prof. Wang is also extremely kind, patient and understanding. He genuinely cares about the well-being of his students and graciously offers his support. I simply could not have hoped for a better advisor and friend. My Master's journey would not have been as joyful without the fruitful collaborations with an extraordinarily talented group of mentors. I was lucky to have the opportunity to learn the basics of fundamental research and presentation from him. Most importantly, Dr Ahmed is an amazing presenter from whom I learned how to keep the audience engaged in an impactful research presentation. I am also thankful to Prof. Gu. I first discovered an interest in optimization theory and machine learning during one of the discussion meetings with him. He always proposed brilliant ideas and was extremely knowledgeable in relation to all the intricate technical details of the proofs. He has been a wonderful source of help and support to me during my research journey. He has been extremely kind and supportive, always ensuring that I knew I could on his help. I will always be grateful and indebted for this. Last but definitely not least, I owe this research to my beloved family, my husband, Mohammed Obaid Ahmed, and my mom, Bilquis Begum, without whom I could not have completed this journey without their help, support, encouragement and sacrifice. I have been extremely lucky, Alhamdulillah (being thankful to the Almighty the most for giving me these two amazing persons in my life along with my twin infants who've added all the happiness I needed to rejoice every time I felt discouraged). This thesis is dedicated to my husband, mother, and my little twins.

List of Publications

Conference Paper

- Elastic Optimized Edge Federated Learning

Journal Paper

- Elastic Optimization for Stragglers in Edge Federated Learning (In Progress)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Overview

This chapter commences with the preliminaries in section 1.1. The problem context and motivation for this research is presented in section 1.2. The overall aim of the thesis followed by the research questions are covered in section 1.3. Limitations are presented in section 1.4 followed by contributions in section 1.5. Finally, the structure of the thesis is outlined in section 1.6.

The sheer amount of data generated due to big data has resulted in it being centralized at a single place or in an isolated centralized database. For distributed machine learning (DML), the typical assumption is a client-server architecture where one party acts as a server which is responsible for data accumulation and the other party acts as a client who is responsible for sending data to the server for aggregation and communicates with the server for any type of predictive analysis [73, 86, 28].Further, the ever-growing adoption of the Internet of Things (IoT) for smart and intelligent applications has increased the usage of IoT mobile/edge devices [37, 126, 164]. Over the last few decades, machine learning (ML) inference on mobile edge devices is possible due to the growing computational capabilities of the edge devices [66, 200, 147]. According to the Forbes 2020 report, "The constant increase in data processing speeds and bandwidth, the non-stop invention of new tools for creating, sharing, and consuming data, and the steady addition of new data creators and consumers around the world, ensure that data growth continues unabated. Data begets more data in a constant virtuous cycle." It is evident that, the data generated by smart devices such as IoT devices, smart phones and various edge devices that can be included in edge computing, today, is increasing enormously and to obtain meaningful information from this data, different technologies, such as artificial intelligence [173, 209], ML [206, 111, 183], data analytics [129, 161, 143], and data science [45, 92, 44], are used. The era of mobile or edge computing has made it possible for edge devices to train ML models. However, the issue of data privacy has always been a critical point of concern [18, 148, 53]. In recent years, the General

Data Protection Regulation (GDPR) [175] has made strict security and privacy compliance compulsory for all the organizations across Europe. GDPR is the world's most stringent privacy law. With the introduction of GDPR, people are more aware of data privacy and its regulations [163, 140, 139]. Traditionally, to start any ML task, we need data in silos. The owners have to make their data available for research which is often of great concern, especially when it comes to private and sensitive data [192, 79, 132].]. The moment data leaves its place of origin, there is no guarantee of its complete safety and that data breaches will not occur. Many studies in literature have focused on secure data storage, data protection in transit, the owner's regulations over their data, to list a few [195, 52, 190]. In a nutshell, data has to leave the owners and reside in a data repository which is a significant privacy concern all over the world. Therefore, to avoid privacy concerns, a new privacy-preserving technology was introduced, by Google known as federated learning (FL) where machine learning is undertaken at the edge devices with their local data thereby avoiding the need to keep data in warehouse [128, 62, 51]. Hence, it is not possible for anyone to see the data on which the models are trained which makes collaborative learning much simpler and more reliable. Consecutively, FL can be thought of as an ML application to support privacy in edge computing, which includes all the data generating edge devices. However, FL has certain characteristics such as system and statistical heterogeneity which can cause obstacles in its real-time implementation. In particular, in edge federated learning, where FL is implemented in edge computing, the two heterogeneities mentioned above can result in the emergence of stragglers. Stragglers are slow devices which delay the communication of their local training results with the edge server. Hence, stragglers can degrade model performance and can affect accuracy, thereby decreasing model utility. In edge federated learning, there are millions of edge devices available for FL training. These devices communicate with the edge servers for model aggregation at the end of each training round. The training iterations stop once target accuracy is achieved. One critical aspect in FL training is to ensure the model training is efficient by reducing the number of communication rounds while achieving target accuracy. This is only possible when the local objectives of the edge clients are optimized from the heterogeneities present. The recent literature suggests two reasons for the emergence of straggles, firstly, the large number of parameters from high dimensional edge devices data; and secondly, statistical local objective divergence due to the partial participation of edge devices. These reasons cause computation and communication issues with the edge server as large parameters result in a computation burden whereas statistical divergence results in a drop in ac- curacy and disturbs the regression analysis. To deal with the large number of parameters or features and statistical heterogeneity, which causes stragglers, various techniques have been used in the literature. In the recent literature, many studies such as [81], [59], [202], [84], [13], [49], [221], [85] focus on making federated learning efficient.

Some of techniques involved are discussed in the following sub-sections.

1.1 Federated Learning

FL was first coined in 2016 [96]. McMahan et al. introduced a new way of obtaining decentralized machine learning without data centralization

Typically, the steps involved in the FL are as follows:

- Global model parameters are initialized at the server.
- Global parameters are sent or broadcasted by the server to all the participating clients.
- Clients perform local training by using the downloaded global parameters for a certain number of local iterations.
- Local model parameters are sent to the server by the clients.
- The server obtains a global model after aggregating the parameters received by all the clients. The newly obtained global model parameters are again shared with the clients in the subsequent training rounds until a target accuracy is reached.

Since its inception, FL has been a trending area of research and has attracted significant interest from both research and applied perspectives. Although, there has been a lot of work on implementing and integrating FL in different frameworks, such as mobile edge computing, wireless networks, etc., it has received little attention in relation to making it efficient in the presence of stragglers which delays server aggregation. This motivates us to research the straggler effect in FL settings. The massive distribution of clients with varying heterogeneities can cause severe performance degradation if the stragglers are not handled properly. Our objective is to mitigate stragglers by adopting a hierarchical structure to overcome the rigid topology of baseline FedAvg.

1.2 Algorithm for federated learning

Federated averaging (FedAvg) is the algorithm designed by [96] for FL. FedAvg is considered the baseline algorithm for FL and uses stochastic gradient de- scent (SGD) as the local optimizer. FedAvg follows a synchronous approach where global model aggregation is performed once all the participating de- vices have submitted their local updated results to the server.

The algorithm for FL is as follows:

1.3 Motivation/Research Problem

Currently, the benchmark algorithm for FL, federated averaging (FedAvg), requires all the participating edge devices to perform the same number of training rounds regardless of their system and statistical heterogeneity. These heterogeneities result in stragglers who cannot complete their all the training rounds

```
Algorithm 1: Federated Averaging
Input: B, N, E, $\eta$
Output: federated trained global model
while target accuracy not achieved do
    Global model parameters broadcast to edge devices;
    Local model training at each client;
    if complete local training after E updates then
        send the updated parameters to the edge server;
    end
    if local updates received by the edge server then
        perform aggregation using equation(3);
    end
end
```

thereby delaying the global model aggregation. Stragglers have been a topic of interest in distributed systems from for the past few years. The interest of in stragglers in FL grew quite quickly, as shown in the recent literature. We are interested in edge federated learning where FL is an application in edge computing. In edge federated learning, the straggler issue can degrade model performance as there are millions of participating devices which are uncertain about their prolonged participation as well as complete time dedication for model training. System heterogeneity arises due to the different computational capabilities of the clients with different system configurations. Similarly, statistical heterogeneity is due to high dimensional data with a large number of features or parameters. For instance, the images stacked on top of each other form the video where the pixel values stored could be very complex. Hence, the processing time for this kind of data also requires the devices to completely dedicate their system resources. Any limitation in system resources or complexity in data can result in stragglers, thereby decreasing model utility. A large number of parameters from high dimensional data has been a statistical problem in the literature, where the use of efficient methods to identify the appropriate required features was adopted [55, 56].]. Many of the deep neural networks used for federated model training have millions of parameters due to high data dimensionality [3],[160], [4].This large number of parameters makes it challenging for model training. Deep learning is often helpful in dealing with feature extraction so that the model with the optimal features is selected with a feasible number of parameters [151]. The work in [98] discusses client drift due to the high-dimensional data together with few local steps in the model training which hurts global model convergence. Further,

high data dimensionality of model parameters causes communication latency between the base station and the edge devices in mobile edge computing [118]. In [153],the clients are grouped as per their data distributions and then their individual model is obtained from each group. The benefit of undertaking this clustered approach for model aggregation is that the high dimensional model

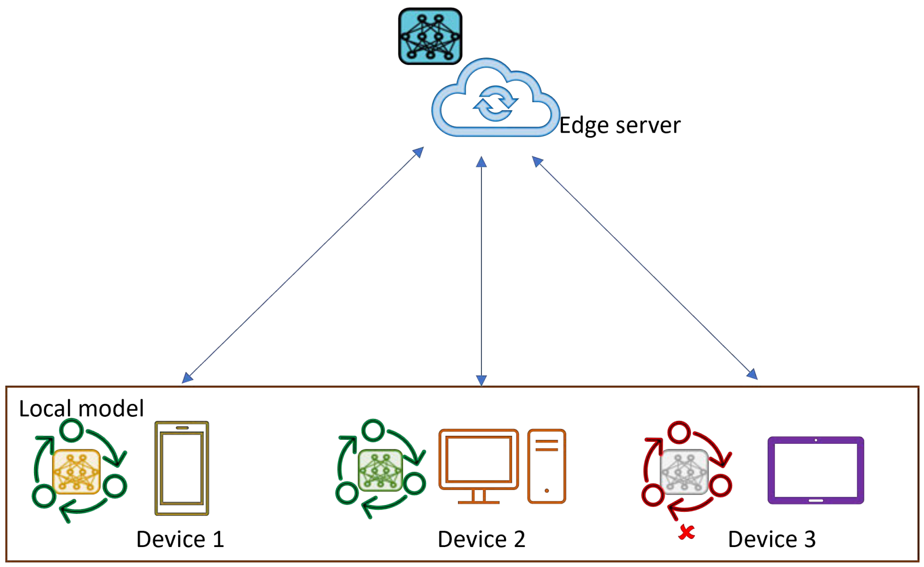


Figure 1.1: Stragglers In Federated Learning

parameters from clients which belong to the other group cannot reach another distribution group, thereby improving model training.

As can be seen in 1.1, there are three edge devices performing local training at the edge. Device 1 is a mobile phone, device 2 is a PC and device 3 is a tablet. Device 1 and device 2 can perform specified local iterations and can update the edge server with their results. But, device 3 has failed to respond to the edge server. Hence, device 3 is a straggling device, which is caused by differences with respect to the computational capabilities compared to the other two devices and the variances in their data when incomplete results are shared after the specified number of iterations.

1.4 Research Aims

1.5 Objective

The research objective of this thesis is to introduce an optimization algorithm that can mitigate stragglers due to data heterogeneity. This algorithm can assist in the utilization of data from stragglers which is other- wise missed from training by ignoring them, as in FedAvg. Hence, an elastic optimization algorithm can help researchers to better analyse straggler mitigation and utilize the potential of straggler data in obtaining target accuracy.

From this principal research objective, the following research objectives are identified:

- to define federated learning (FL) and its characteristics;
- to give an overview of the various FL algorithms for stragglers in the literature and to investigate how they differ from each other;
- to introduce an elastic optimization algorithm which is different to any of the approaches in the literature for straggler mitigation;
- to evaluate the proposed algorithm.

1.6 Research Contribution

Existing work has investigated and developed various distributed optimization algorithms to solve the straggler problem which causes communication inefficiency and accuracy degradation. However, the following crucial questions remain unanswered:

- What could be the reasons for straggler emergence in edge federated learning. Is there an algorithm as simple as FedAvg which can solve this problem and how can we compare this algorithm with the benchmarks?
- FedAvg generally doesn't consider stragglers and ignores the valuable information stragglers can provide which may eventually improve regression analysis. Is there a distributed algorithm which is as simple as FedAvg but works better at incorporating straggler information. How can the re-designed FedAvg achieve better accuracy and ensure communication is more efficient?
- How much better can the revised FedAvg perform in terms of accuracy and communication rounds? Can we compare the new distributed algorithm with benchmarks such as FedAvg and FedProx?

This research aims at addressing the above thought-provoking questions about stragglers in edge federated learning. Our contributions are summarized in the next section.

- We formulate stragglers as the distributed optimization problem in edge federated learning where the optimization involves the minimization of the the local objective loss function for all the edge clients participating in the federated training. By formulating a distributed optimization for stragglers, we are trying to achieve faster computation with better accuracy over less communication rounds.
- We propose the Elastic Optimized Edge Federated Learning (FedEN) approach to deal with stragglers due to data heterogeneity which is a result of today's high dimensional data and partial edge device participation. FedEN mitigates stragglers by dealing with the large number of features with the help of lasso penalization and generates sparse efficient models.

Furthermore, the edge devices stay near to the local optima with the help of ridge penalization thereby restricting the strict feature elimination of lasso. Hence, the elastic optimization in edge federated learning gives an optimal balance between lasso and ridge, thereby benefiting edge federated learning with the elimination of stragglers.

- Finally, our extensive experiments performed for image classification using FedEN prove that our work achieves substantially high accuracy and lower loss compared to the benchmark algorithms.

1.7 Research Questions

The following research questions aim to address the issue of stragglers being ignored by the benchmark algorithms

- What is the model accuracy when stragglers are incorporated in edge federated learning?
- How can we improve training performance in the presence of stragglers?

The motivation for research question 1 is because stragglers degrade model performance. The aim is to analyze the performance of the image classification model in the presence of stragglers in model training. Further, we examine how the benchmark algorithms FedAvg and FedProx behave when stragglers are incorporated in training. Research question 2 deals with the performance optimization of federated training by considering the elastic optimization in the local objectives of the participating edge devices.

1.8 Publications

Conference Paper

- Elastic Optimized Edge Federated Learning

Journal Paper

- Elastic Optimization for Stragglers in Edge Federated Learning (In Progress)

1.9 Thesis composition

This section provides an outline of the thesis. In Chapter 2, we discuss the background information required to understand the concepts and algorithms in further chapters. In general, chapter 2 covers all the concepts of machine learning and distributed machine learning, the optimization algorithms in the literature, the regularization techniques for distributed optimization, the basics of federated learning and gives a brief overview of the benchmark federated averaging algorithm. Finally, edge federated learning is briefly discussed. In Chapter

3, we discuss the stragglers in the literature and how they are mitigated. We discuss the reasons for stragglers in distributed systems followed by a discussion of the emergence of stragglers in federated learning and edge federated learning. In Chapter 4, we discuss the pro- posed distributed edge federated learning framework, Elastic Optimized Edge Federated Learning algorithm (FedEN) and its convergence analysis. We briefly overview the data collection as well as data heterogeneity and convergence properties. We then describe the methodology of the proposed research which involves logistic regression for image classification and optimization with stochastic gradient descent. Next, we discuss the emergence of stragglers as the distributed optimization problem followed by the proposed regularization technique used for straggler optimization. Finally, discuss the theoretical assumptions and theorems for FedEN convergence analysis. In chapter 5, we discuss the datasets used for the experiments and how data partitioning is performed for federated training. We then discuss heterogeneity and the hyper-parameters used for training. Next, we discuss the implementation details including the loss function and optimizer used for training. We describe each implementation module for the server and for the clients involved in the training process and obtain results for 200 training rounds. Finally, we compare the test accuracy for the CIFAR-10 and MNIST datasets for 10% and 20% of stragglers in the training for over 200 communication rounds.

Chapter 2

Background

In this section, we discuss the preliminary information required to understand the concepts related to machine learning (ML) and federated learning (FL). We start with the background knowledge of ML followed by distributed machine learning, deep learning, convolutional neural networks in ML model training [26, 208, 145]. Next, we provide a generic explanation of FL, its characteristics and types, followed by a discussion of the benchmark algorithm, federated averaging.

2.1 Machine learning

Turing proposed the idea of machine intelligence in 1936. Since then, the interest in machine learning, to create intelligent models for better decision making has grown rapidly [41, 113, 144, 78]. ML is a core part of several technologies such as 6G, blockchain, autonomous vehicles to name a few. The majority of computations inmachine learning are based on the basic transformations for matrices or tensors. Finding a way to optimize these operations has been an area of research interest in the literature. As a result of this, many techniques such as high-performance computing [21] have been an active area for research in academia as well as in industry.

ML is a very im- portant concept in deep leaning as well as artificial intelligence. The combination of machine learning and artificial intelligence has led to the development of robot-nurses, self-driving cars etc. The insights obtained from ML can help in better decision making in almost all fields, such as healthcare, manufacturing and automation. Machine learning or deep learning are sub-fields of artificial intelligence. Deep learning, however, is a sub-field of machine learning. Neural networks, on the other hand, are a sub-field of deep learning. The main difference between machine learning and deep learning is how their algorithm learns. Generally, deep learning is used for feature extraction, reducing the manual human effort of feature extraction, whereas machine learning is more dependent on human intervention for feature extraction. Machine learning comprises the following stages : 1) data collection; 2)

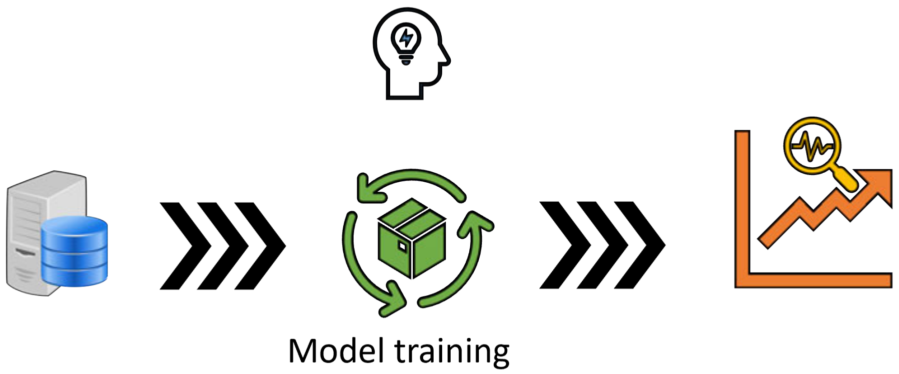


Figure 2.1: Machine learning from centralized data

pre-processing; 3) feature extraction; 4) training; 5) testing the model for its accuracy; 6) re-training with updated parameters or adjusted features., A machine learning algorithm comprises the following three parts: 1)decision making - the algorithm has to make decision based on what has been set for the algorithm (classification, for instance) and will give an output given the labelled or unlabeled data to process and provide an estimate; 2) error function: this gives an estimate on the accuracy of the model predictions. Error functions usually help to determine if the model or algorithm has better accuracy over the test data; 3) model process: When the machine learning model is trained, it fits the data points in the training set. Further, when the predicted value fits the data points in the training set, then the model is said to be generalized. When only the training set is identified by the model, the model is said to be over fitted and it does not have a better generalization error. Therefore, sometimes to avoid overfitting, regularization is used which induces bias while training and helps improve the generalization error. Therefore, regularization plays an important role in reducing overfitting. Two phases are involved in machine learning: 1) training the model; and 2) testing the model. During the training phase, the collected data is fed to the model and hyper-parameters are adjusted either manually or using cross- validation or other techniques. The output of the training phase is a trained model which is ready for the testing or evaluation phase. During the model testing phase, the model is deployed to check its generalizability. The training phase is computationally difficult while the testing phase is easier and requires less computing power.

2.1.1 Distributed Machine learning

Distributed machine learning (DML), as the name implies, consists of multiple nodes geographically in different locations but connected with each other. DML improves accuracy as it reduces the need for centralized training and enables the incorporation of information from multiple worker nodes. To enable faster processing time, computation can be performed with other devices. One of the most important aspects when discussing distributed machine learning relates to accuracy goals and reducing the communication and computation overheads. However, when communication and computation delays increase with the machine learning problem, then subsequently accuracy also increases. To reduce the delays associated with communication and computation in distributed machine learning, parallelism is widely adopted to make the process flexible for model training. The following several techniques are used for parallel computation and communication between workers:

Bulk Synchronous Parallel The bulk synchronous parallel (BSP) model is a simple technique where workers communicate effectively and it is synchronous in nature. It is a parallel computing technique comprising a series of supersteps. Each superstep has processes that run the same code in a parallel manner and later the messages are sent to other processes. At the completion of a superstep, all messages have already been sent to the processes. At the end of a superstep, a barrier for synchronization ensures that all computations are done and messages are transmitted. As soon as the next superstep starts, all the messages are then delivered to the processes. This whole process is iteratively executed starting from processing and delivering at the start of next superstep. BSP has been extensively used in many environments. For instance, MPI-2 mimics the BSP technique including one-sided communication along with barrier synchronization. However, the global barrier for synchronization can result in stragglers since some workers with limited resources can delay the computation or communication process.

Asynchronous Parallel To overcome the growing issue of stragglers in BSP, asynchronous parallel (ASP) processing was developed to ensure the participating worker gains faster access to messages compared to BSP. Workers with fast computational capabilities can move forward without waiting for the stragglers. However, since waiting for stragglers is not a consideration in ASP, stale gradient computation is involved in the process. Stale gradient computation increases the computation time and decreases the accuracy of the global model due to the incorporation of stale gradients.

Stale Synchronous Parallel A stale synchronous parallel (SSP) model was developed since neither BSP nor ASP can work better with different algorithms for better accuracy with reduces communication and computation delays. SSP works better than BSP by allowing fast workers to work by a fixed number of steps ahead. This is known as gradient bounded staleness. Bounded staleness ensures that there is a bound on the staleness that a model can bear and the amount of trade-off that can be made in relation to accuracy and delays associated with communication and the computation of resource-constrained workers. SSP greatly reduces stragglers because, on completion of their process, workers communicate and proceed with the next round thereby improving training efficiently. However, as staleness increases over time, accuracy reduces dramatically.

2.1.2 Machine models

Parallelism in machine learning can be achieved in two ways: 1) model parallelism; and 2) data parallelism.

Model Parallelism As the name indicates, model parallelism divides model parts among participating worker nodes in machine learning. Each worker has an entire copy of the data set to be processed. This entire data set is used by different workers to train different part of the models. The actual model is a summation of all model parts aggregated into one global model.

Data Parallelism In data parallelism, training data is split across many devices in the training process. Each worker has a copy of the whole model for training. Each worker node applies the same algorithm to the different data subsets. This works better when the data each worker hold come from the same or a global distribution. That is, data parallelism assumes data to be independent and identical (IID) for every data sample each worker holds. Hence, federated learning does not allow raw communications, rather federated learning is a distributed machine learning approach.

2.1.3 Deep learning

Deep learning is a making learning method based on artificial neural networks. Deep learning revolutionized the world with its growing development in all fields, especially in image and text classification. Traditionally, in the machine learning process, features are manually extracted which is the most crucial step before model training. The better the features extracted, the better the accuracy of the trained model. More recently, deep learning has been a hot area of interest for automatic feature extraction and has been widely used in the literature for many machine learning applications. Further, deep learning does not require manual feature engineering and takes raw data as the input. Deep learning works with both structured or un-structured data. Deep learning algorithms can easily handle complex operations efficiently which machine learning cannot. Further, with an increase in training data, the performance of machine learning algorithms decreases, whereas deep learning can easily handle a large amount of data and can efficiently extract features.

2.1.4 Convolutional neural networks

Convolutional neural networks (CNNs) are commonly used for image classification [104, 124, 71]. The default assumption for CNNs is that the input is images which makes the forward function more efficient and reduces the parameters. CNNs requires less pre-processing and can perform better feature extraction than usual neural networks. CNNs, in general, use convolution filters which slide over whole image and extract the features [191, 193, 162, 166]. Three layers are used in the CNN architecture: convolutional layers, pooling layers, fully-connected layers. Convolution in mathematics is an operation on two functions which describes how the shape of one function is modified by the other. The result of this operation yields a feature or activation map. The next layer, the pooling layer, performs downsampling. There are two types of pooling - max pooling and average pooling. Max pooling, as the name indicates, takes the maximum value of the view of the convolution filter, preserving the detected features, whereas average pooling downsamples the activation map by averaging the view of the current value. The last layer of CNNs is the fully connected layer , which works on the flattened input where each input is connected to all neurons in the network.

Convolutional layer

A convolutional layer, as its name indicates, is based on a mathematical operator which is called a convolution. For a any given two functions, f and g such that $f : Z^2 \to R$ and so $g : Z^2 \to R$, the convolution operation between the two at a point, p, such that $p \in Z^2$ can be defined as follows:

$$(f * g)(p_1, P_2) = \sum_{n_1 \in Z} \sum_{n_2 \in Z} f(p_1 - n_1, p_2 - n_2) g(n_1, n_2) \tag{2.1}$$

f represents the input from the preceding layer for a convolution and g is the convolutional kernel that is used to move through the convolution. Usually, the output feature map is smaller than the input feature map as a result of the convolution operation. Further, the output feature map is the average of the input from f with the convolution kernel g which is obtained when the kernel transverses over the input feature map. This traversal creates an output map which is then passed to the next layer. Depending on the size of the kernel as well as how the boundaries of the input feature map are considered, the result is the size of the output feature map.

Pooling Layers

Usually after the convolutional layers, pooling layers are used. There are two types of pooling layers: 1) max pooling; and 2) average pooling. Max pooling extracts the maximum value from each neuron's local neighbourhood, whereas average pooling extracts the average value from the neighbourhood of local neurons. Pooling helps in reducing the number of parameters for the subsequent rounds of the network. The pooling layer operates independently on every input and resizes the input using the MAX operation. The most common form of convolutional layer is a pooling layer with filters of size 2x2 with a stride of 2 downsamples every depth slice, discarding 75% of the activations.

Fully Connected Layers

In the fully connected layer, neurons are connected to all the activations in the previous layer. These layers are placed at the end of the convolutional layers. Every neuron has a trainable bias which is associated with it. This is represented by the following equation:

$$y = \sum Wx + bias \tag{2.2}$$

where W is the weight of the input neuron and b is the bias of the neuron.

Deep Learning - Platforms There are many frameworks currently used for deep learning. They can be listed as follows:

- TensorFlow : TensorFlow is an open source platform available for machine learning. It consists of many libraries to support many deep learning applications. TensorFlow has many built-in modules and pre-trained models which can be used directly to be incorporated in the existing works. It can be programmed using Python. Since Python is an easy programming language providing high level abstractions, it allows flexible programming for tensorflow. The main reason to use TensorFlow is that it provides the abstraction of modules which does not require either a technical or non-technical person to get in detailed description such that a person can just focus on the application logic. TensorFlow can be used for heterogeneous large-scale environments. It can be regarded as a platform for many research areas such as computer vision, natural language processing, information systems.

- PyTorch : This is another open source deep learning framework which is used extensively in re- search. PyTorch provides high-level features to support production deployment. Recently, with the introduction of distributed machine learning and gradient quantization, PyTorch has been widely adopted in academia and industry. PyTorch supports the asynchronous execution of tasks which can optimize the training. It is similar to the NumPy operation and accelerates the training process using a graphical processing unit. PyTorch supports more than 200 mathematical operations. The key features include the following : 1) the availability of TorchScript which enables users to move between different modes with easy to use and flexible operations , dynamic graph computation which allows the participants to engage in dynamic network behaviour, allowing changes as per their network conditions on the fly, differentiation auto-calculation which ensures that the differentiation of gradients is done automatically, Python packages which ensures the availability of multiple libraries such as NumPy, SciPy and many more.

- MXNet: MXNet is an open-source deep learning framework that allows deep neural networks to be defined, trained, and deployed on a wide array of devices., It is a deep learning framework developed using various program- ming approaches. The programming approaches supported by MXNET include Python, C++, Perl, R, to name a few. A unique advantage of building a model using MXNET is due to the fact that MXNET models are portable and can fit in a small memory. Therefore, once the model is trained and testing using the MXNET framework, it can easily work on mobile devices. Due to its scalability, it is used by many companies including Amazon to provide deep learning services.

- Keras is a Python framework to perform deep learning model training. With Keras, it is straightforward to define a neural network structure and undertake its training. It provides high- level APIs that can work with backends such as Theono, TensorFlow etc.
- DL4J: Eclipse DeepLearning4j (DL4J) is the open source deep learn- ing framework with many distributed libraries written for Java and Scala. DLJ4 enables CPUs and GPUs to be used in distributed environments. It allows a combination of neural nets such as en- coders and CovNets as per the distributed requirement. DL4J has a large number of parameters which can be adjusted during deep learning training.

2.2 Optimizers in Deep Learning

An optimizer is an important part of a neural network which helps in determining the appropriate weights required for accurate prediction. This can be achieved by minimizing the local objective function of the devices in training neural networks. Depending on the output weights achieved, optimizers can be categorized into the following different types:

- Gradient Descent
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent
- Momentum based gradient descent
- AdaGrad
- Nesterov Accelerated Gradient
- RMSProp

2.2.1 Gradient Descent:

Gradient descent is one of the iterative optimiza- tion algorithms to train a neural network. Gradient descent works by minimizing the objective function by moving into a negative direction of the computed gradients so that the steepest point is obtained and optima is reached. The negative direction is computed by taking the derivative of the slope of the objective function at a given point. For instance, let the objective function be denoted as $f(\omega)$ with respect to the model parameter ω. Therefore, the derivative for $f(\omega)$ is denoted by $f^{`}(\omega)$ which can be formulated as follows:

$$\begin{aligned} f^{`}(\omega) &= \frac{df(\omega)}{d\omega} \\ &= \lim_{h_k \to 0} \frac{(\omega + h_k) - \omega}{h_k} \end{aligned} \tag{2.3}$$

From the equation above, it is worth mentioning some of the key points for $f^{`}(\omega) > 0$ which are as follows:

- the function is growing locally upwards
- the function is growing locally downwards
- the function has a local stationary point

Therefore, gradient descent can be formulated as follows:

$$\omega_a = \omega_{a-1} - \eta \nabla f(\omega_{a-1}), a \in R \tag{2.4}$$

where, ω is the model parameter and η is the learning rate.

For multivariate functions, the above equation can be written with respect to points x,y. Therefore, the multivariate function for any point x, y is given as follows:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h_k \to 0} \frac{(x, y + h_k) - f(x)}{h_k} \tag{2.5}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{h_k \to 0} \frac{(x, y + h_k) - f(y)}{h_k} \tag{2.6}$$

For a multivariate function, there are infinite possibilities of the gradient direction. Therefore, the directional derivative for a vector for a function $f(x, y)$ of the form $v =< a, b >$ can be expressed as follows:

$$f(x,y)_v = \lim_{h_k \to 0} \frac{f(x + ah_k, y + bh_k) - f(x,y)}{h_k} \tag{2.7}$$

Therefore, the gradient descent is given as follows:

$$< x_a, y_b >=< x_{a-1}, y_{a-1} > -\eta \nabla f(x_{a-1}, y_{a-1}) \tag{2.8}$$

And, also the gradient update is given by

$$x_a = x_{a-1} - \eta \nabla f(x_{a-1}) \tag{2.9}$$

2.2.2 Batch Gradient Descent:

The computation in batch gradient descent is carried out to obtain the gradient of the objective function with respect to the model parameters. It obtains the training samples for a particular iteration and processes it as a batch at a time. Moreover, the training can be computationally expensive when the batch size is larger thereby increasing the processing time. Therefore, when the number of samples in the batch is larger, batch gradient descent is not an option. Instead, stochastic gradient descent or mini-batch gradient descent is an alternative.

Pseudocode for batch gradient descent:

- Initialize the model weights vector
- Until the termination condition is met, take batch of sample for training
- Update the model parameter after each batch training

2.2.3 Stochastic Gradient Descent

Different from batch gradient descent (iterative), stochastic gradient descent (SGD) is an incremental approach to moving in negative direction of the gradients to reach the steepest point. SGD updates the model parameter, ω, at every step of the iteration. Hence, the computation time is greatly reduced in SGD. However, when the training samples are larger, updating parameters per iteration causes overhead. Compared to batch gradient descent, SGD results in higher variances as it computes the noisy gradients for the input samples. SGD also computes the stochastic approximation of gradient descent optimization. The choice of learning rate plays a crucial role in the working of SGD. A smaller learning rate makes the variance larger and delays convergence, whereas a larger learning rate does not reach the local optima as larger steps are taken by the algorithm and minima might be missed due to these larger steps. So, in general terms, a constant learning rate is used to reach half of the convergence and then the learning rate is decreased by some constant. This is also called learning rate decay where we start with a small constant learning rate and then eventually drops it by some constant to achieve stable convergence.

Pseudocode for stochastic gradient descent:

- Initialize the model weights vector
- Until the termination condition is met, take each sample for training
- Update each model parameter per iteration

2.2.4 Mini-batch gradient descent:

Mini-batch gradient descent takes a subset or mini-batches from the samples to compute gradients per iteration. It is often a choice to use mini-batch gradient descent when the training set has a larger number of parameters as it is efficient to use mini batches of data samples instead of the whole batch in each iteration or each parameter per iteration.

Pseudocode for mini-gradient descent:

- Initialize the model weights vector
- Until the termination condition is met, take each sample for training
- Update each model parameter per iteration

2.3 Regularization Methods

Regularization is an old technique but is powerful when dealing with statistical problems in data. More recently, statistical problems due to high dimensional data generated by edge devices have become more prevalent.

to achieve a stable classification or regression model from high dimensional data, penalization methods, such as L_1 norm or L_2 norm have been adopted in the literature. Imposing such regularization methods reduces generalization error and improves prediction accuracy. L_1 or L_2 or both together work to eliminate the redundant features and can shrink the parameters for induce smoothness in the objective function. The basic foundations of such regularization norms are detailed in the following section. Different regularization techniques includes the following:

- Lasso Regression
- Ridge Regression
- Elastic Net Regularization

The aforementioned regularization methods differ in the way they impose a penalty on the model coefficients. Lasso(L_1) is well known to induce sparsity thereby selecting appropriate features for the model. Ridge($L2$) is responsible for keeping all the features but tries to shrink them.

2.3.1 Lasso

Lasso is an abbreviation for least absolute shrinkage and selection operator. It is also known as L_1 norm or L_1 regularization. Lasso regression is a linear regression which shrinks the coefficients to a central point. Lasso adds a penalty which is equal to the absolute value of magnitude of the model coefficients. Therefore, lasso produces a sparse model by eliminating some of the coefficients which are made zero. [154] explains the lasso problem which is also known as l_1 penalized linear regression. Lasso estimates for any predictor matrix $A \in R^{nxp}$, with the predictor, y, and tuning parameter λ is given as follows

$$\beta^{‘} \in argmin\frac{1}{2}||y - A\beta||_2^2 + \lambda||\beta|| \tag{2.10}$$

When the rank of predictor matrix A, rank(A) = p, the lasso solution is unique. When the rank is less than p, the solution is not unique as the objective function is non-convex due to the presence of multiple local minima. When the data is high-dimensional, lasso is quite a popular tool since it induces sparsity and produces sparse models. High dimensional data occurs where the features are larger than the number of observations in the data set. The number of features removed or set to zero by lasso all depends on the value of the tuning parameter, λ. If the value of the tuning parameter in equation 2.10 is zero (= 0 then it is the linear regression estimate equivalent. The inference is due to fact that when the data is high-dimensional, the linear regression estimate is never unique or uniquely identified. Let us consider the following conditions to explain the unique lasso solution:

Corollary 1: The lasso problem as per equation 2.10 for any output predictor, y, predictor matrix, A, and the tuning parameter, λ has the given following properties:

- Multiple solutions exist or a unique solution is possible
- For every lasso solution there exists $A\beta^{`}$
- Every lasso solution has same l_1 norm.

With the first property of corollary 1, since $f(x)$ is strictly convex and if two lasso solutions $\beta_1^{`}$ and $\beta_2^{`}$ are different from each other, we know $\alpha\beta_1^{`} + (1-\alpha)\beta_2^{`}$ is a solution where $0 < \alpha < 1$. This gives multiple solutions for lasso.

With the second property of lasso, if two lasso solutions$(\alpha\beta_1^{`} + (1-\alpha)\beta_2^{`})$ have different fitted values then for any k minimum value obtained for any two lasso solutions we have,

$$\begin{aligned} 1/2||y - A(\alpha\beta_1^{`} + (1-\alpha)\beta_2^{`})||_2^2 + \lambda||\alpha\beta_1^{`} + (1-\alpha)\beta_2^{`}||_1 \\ < \alpha * k + (1-\alpha) * k = k \end{aligned} \tag{2.11}$$

where $\beta_1^{`}$ and $\beta_2^{`}$ are two lasso solutions for strictly convex function, $f(x)$.

With the third property of lasso, if $\alpha\beta_1^{`} + (1-\alpha)\beta_2^{`}$ have the same fitted values , then they have same square error. If the tuning parameter $\lambda > 0$, they must also have same l_1 norm.

Karush-Kuhn-Tucker (KKT) Optimality conditions: KKT conditions for lasso solution can be defined as follows:

$$A^T(y = A\beta^{`}) = \lambda\Gamma \tag{2.12}$$

$$\Gamma_i \in \begin{cases} sign(\beta^{`j}), & \text{if } \beta^{`j} \neq 0 \\ [-1,1], & \text{if} \beta^{`j} = 0, forj = 1, 2, ..., p. \end{cases} \tag{2.13}$$

In the above equation, Γ is the sub-gradient of a function, $f(x)$, which is equivalent of the l_1 norm of x.

We define the equicorrelation set and signs as follows:

$$\begin{aligned} \xi = \{i \in \{1,2,...,p\} : |A_i^T(-\nabla f)(A\beta^{`})| = \lambda\}, \\ s = sign(A_\xi^T(-\nabla f)(A\beta^{`})) \end{aligned} \tag{2.14}$$

The equicorrelation set is so named because for the predictor variable, y, predictor matrix, A is standardized and the equi-correlation matrix consists of variables that have the maximum correlation with respect to the residual.

The unique lasso fit for $A\beta^{`} = A_\xi$ and any lasso solution $\beta^{`}$ can be written as follows:

$$A\beta^{`} = A_\xi(A_\xi)^+(y - (A_\xi^T)\lambda s) \tag{2.15}$$

Similarly,

$$\begin{aligned} \beta^{`}_{-\xi} = 0 \\ \beta^{`}_{+\xi} = (A_\xi)^+(y - (A_\xi^T)\lambda s) + b_l \end{aligned} \tag{2.16}$$

where b_l produces a lasso solution such that $b_l \in null(A_\xi)$ satisfying the following conditions :

$$s_j.[(A_\xi)^+(y - (A_\xi^T)\lambda s)_j + b_{lj}] \geq 0 \tag{2.17}$$

The above equation is true when the value of j is such that $j \in \xi$.

LARS Lasso Algorithm The LARS algorithm calculates the lasso solution path. When the tuning parameter $\lambda = \infty$, the lasso solution is zero. As the parameter tends to decrease, it computes a solution as a piecewise linear continuous function of the parameter. The solution path is the combination of multiple links with each link representing an iteration of the algorithm satisfying KKT optimality conditions. The LARS algorithm assumes that when $rank(A_\xi) = |\xi|$ when computing its solution path. The LARS algorithm for computing the lasso path is as follows:

- Starting with the count, j=0, parameter $^{`} = \infty$, correlation set, ξ, and the sign as s.
- While $\lambda_j > 0$
  - compute the lasso solution as in equation 2.16 and 2.17 at λ_j
  - compute the solution joining time as in equation 2.18 and 2.19
  - compute the solution cross time as in equation 2.20 and 2.21
  - set the next link in the solution path to the maximal value of join and cross time.

We can compute the equicorrelation set at the beginning of j-th iteration of the algorithm as follows:

$$\begin{aligned}\beta^{\text{‘}LARS}(\lambda_j) &= (A_\xi)^+(y - (A_\xi^T)\lambda s) \\ &= k - \lambda_j d\end{aligned} \tag{2.18}$$

where $k = (A_\xi)^+ y$ and $d = (A_\xi)^+(A_\xi^{T+})s$ give a linear representative function of the tuning parameter, λ. The estimate given above can be assumed as the l2 regularization on the variable ξ as follows:

$$\beta_\xi^{\text{‘}LARS}(\lambda_j) = argmin||\beta_\xi^{\text{‘}}|| : \beta_\xi^{\text{‘}} \in argmin||y - (A_\xi^T)^+\lambda_j s - A_\xi\lambda_\xi||_2^2 \tag{2.19}$$

Next, the joining time of a variable is the time when an outside variable from the equicorrelation set attaining a maximal value of the absolute inner product will join the set. We denote this time the $\lambda_{j+1}^{joining}$. When a variable inside the correlation set crosses the coefficient path through zero, we call this as next cross time denoted as $\lambda_{j+1}^{crossing}$.

A simple and easy calculation for the situation where $j \neq \xi$ can be written as follows:

$$time_j^{joining} = A_j^T(y - A_\xi c)/(\pm 1 - A_j^T A_\xi d) \tag{2.20}$$

Following the above joining time, the next joining time can be given as follows:

$$\lambda^P joining_{j+1} = Maximum(time_j^{joining}) \tag{2.21}$$

Similarly, the cross time of a variable is given as follows:

$$time_j^{crossing} = [A_\xi^+ y]_j/[(A_\xi^T A_\xi)^+ s]_j.1[[A_\xi^+ y]/[(A_\xi^T A_\xi)^+ s]] \tag{2.22}$$

Subsequently, the next crossing time for the variable inside the correlation set can be given as follows:

$$\lambda^{crossing_{j+1}} = maximum[t_j^{crossing}] \tag{2.23}$$

Properties of LARS:

2.3.2 Ridge regression

For high-dimensional data and to select the parameters during the training process, ridge regression is more commonly used. Ridge regression is also known as Thikonov regularization. Matrix A can have a least square estimate which is not unique due to the presence of many predictors. Ridge regression has the ability to overcome the criteria of $(A^T A)^{-1}$.

Residual Sum of Squares: Residual Sum of Squares (RSS) measures the amount of variability or variance in the given data of a regression model. Generally, the sum of squares is the term used for statistical analysis. The mathematical way to represent the best fit line is through the sum of squares. RSS represents the difference between the actual regression function investigated and the data set used for model train. RSS for a function $f(x)$ and output variable y is as follows:

$$RSS = \sum_{j=1}^{n} (y^j - f(x_j))^2 \tag{2.24}$$

where y^j is the j-th predicted variable, $f(x)_j$ is prediction of y_j over the entire n data entries.

Least Squares Estimate (LSS): The least square estimate for coefficient β is defined as follows:

$$\beta^{lss} = \arg\min_{\beta} Rss(\beta) \tag{2.25}$$

RSS tries to find the values that best fit over the line for the regression model by making $RSS(\beta$ small. Generally, all the coefficient estimates are shrunk towards zero. Similar to the least squares estimate, ridge regression can be defined as follows:

$$RSS(\beta) = \lambda \sum_{j=1}^{n} \beta_j^2 \tag{2.26}$$

where λ is the tuning parameter which controls the strength of the penalization. the above equation defines the coefficients for ridge regression. Also, the following shows different approaches to the values of the tuning parameter:

- $\lambda = 0$, we can get the estimate of the linear regression
- $\lambda = \infty$, we can get the ridge estimate as zero
- $0 \leq \lambda \leq 1$, we can get a balanced fit with coefficients shrunken near to zero making a smooth function.

The bias and variance of ridge regression depends on the amount of shrinkage done on the tuning parameter as follows:

- As the amount of shrinkage on the tuning parameter increases, the bias in the data also increases
- As the amount of shrinkage on the tuning parameter decreases, the variance also decreases

Therefore, the amount of shrinkage depends on how the tuning parameter is controlled. Larger values of λ cause more shrinkage and a higher bias in the data as we get different estimates for coefficients for different values of λ and smaller values reduce the shrinkage intensity and thereby decrease the variance in data.

2.4 Federated Learning

Federated learning (FL) [96] has been a hot area of research ever since its evolution by Google. It has opened opportunities to perform collaborative distributed machine learning while preserving data privacy. The ideology of FL is to provide on-device training while complying with data privacy issues. For a device to participate in the FL training process, it must be connected to a network, idle and must be charging. Since the number of edge devices (such as IoT devices) are increasing daily, the potential to obtain an accurate global model is no longer just a whim. That is, apart from privacy concerns, FL also exploits the benefits of the computing power of edge devices. The first FL open source platform known as the Federated AI Technology Enabler (FATE) was developed by WeBank.

Some of the benefits of FL over traditional centralized ML are as follows:

- Since data never leaves its source, the risk of data exposure to unauthorized organizations for ML is greatly reduced. Hence, data privacy is a critical aspect of FL for ML at the edge.
- The data being processed at the edge devices reduces the computation and storage efforts otherwise required for centralized training. Hence, edge devices perform local computations with their data and only the results are shared with the server.
- Obtaining a shared and collaborative model trained on data from large number of devices is the fundamental aspect of FL. Hence, the future of ML with FL at the edge can be seen as a promising solution which can enable different organizations to work collectively without compromising their data.

2.4.1 Characteristics of federated learning

This section details the characteristics of FL briefly and explains each one. These characteristics are based on the description of FL in [96], [57], [80], [61].

- Data privacy : Each edge device uses the data produced locally. Hence, data is not stored in a centralized server which preserves privacy to some extent. However, some inference can be performed as in [131, 142, 165], to gain insights from the gradients in transit from the edge device to the server. Hence, there are several other techniques used in combination with FL. One such technique is differential privacy which tries to hide the actual information by adding some noise to the data transmitted [174, 146].
- System heterogeneity : Edge devices participating in FL vary in their system configurations such as storage capabilities and computation resources. This can affect the number of communication rounds the device can perform with the edge server and the amount of data they can process at a time for local model training. Therefore, edge devices with varying

capabilities participating in FL training can have a significant effect on performance.

- Statistical heterogeneity: Apart from system heterogeneities, heterogeneity in data is the other challenge in FL.
- Massive client Distribution:
- Communication limitation:

As can be seen in Figure 2.2, federated training takes place collaboratively, considering one edge layer at a time for simplicity, with an edge server communicating the initial and subsequent updated intermediate models with the edge devices. Once the model parameters are updated by all three edge devices by performing local training, the updated parameters are shared back to server for model aggregation where all three models are aggregated into one global model. This process continues until target accuracy is achieved.

2.4.2 Types of federated learning

Consider N edge devices with their respective data set D. A traditional approach is to collect all the data from N edge devices and utilize the collected centralized data to train an ML model, M_{total}. In FL, a trained global model M_{global} is obtained without total data, D, centralization at the server. Let the performance measure of traditional centralized and federated learning be denoted as ν_{total} and ν_{global}. The performance of the FL model can be formulated as

$$\nu_{total} - \nu_{global} < \phi$$

FL is known to have $\phi-$performance if the difference between traditional centralized and federated trained models is less than a non-negative real number ϕ.

Depending on how the features and data samples are distributed, federated learning is classified into different categories. The classification is as follows: vertical federated learning (VFL), horizontal federated learning (HFL) and federated transfer learning (FTL).

Horizontal FL

When two parties have overlapping data features between them, this type of federated learning is horizontal federated learning. However, they differ in the data samples. This is also known as sample-based federated learning [186]. Figure 2.3, shows an example of horizontal federated learning with three hospitals which specialize in treating cardiac patients in different locations. Each hospital shares the same patient features in their database. In other words, an individual might be patient at any of these hospitals or more than one hospital, that is patients might be different among these hospital with similar features. However,

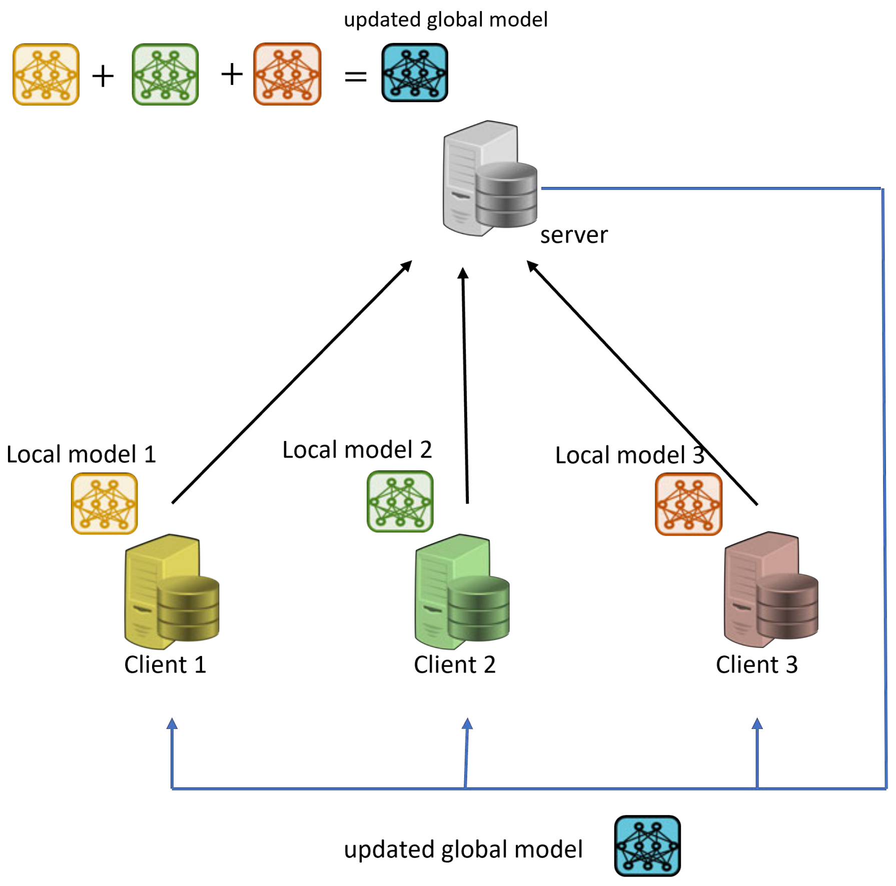


Figure 2.2: Federated Learning Training

the feature space for all the patients is the same. Therefore, these three hospitals can work collaboratively to obtain a global and better predictive model where data features overlap to a great extent. The overall objective of horizontal federated learning is to obtain a globally trained model with or without central server participation for the orchestration of intermediate results. Recent studies such as [30], [33], [214] investigate the horizontal federated learning framework for their experiments.

Traditionally, ML is performed by centralizing the data from different locations [25, 201, 157]. With horizontal federated learning, data are not shared with the server, instead only the gradients or model parameters are shared with the server. The procedure involved in horizontal federated learning involves the following steps:

- Each edge device participates in the federated learning process and obtains the model parameters from the server

- Upon receipt of the model parameters, devices perform local training, returns gradients in encrypted form back to the server
- the server returns updated aggregated model parameters back to the edge devices for further training

For instance, two or three hospitals only share a few patients whose information is in all these hospital's databases. Instead, the features of patients are commonly shared among each hospital's database. Therefore, with horizontal federated learning, the hospitals with the maximum overlap of features help to obtain a global collaborative model which can better work for regression analysis.

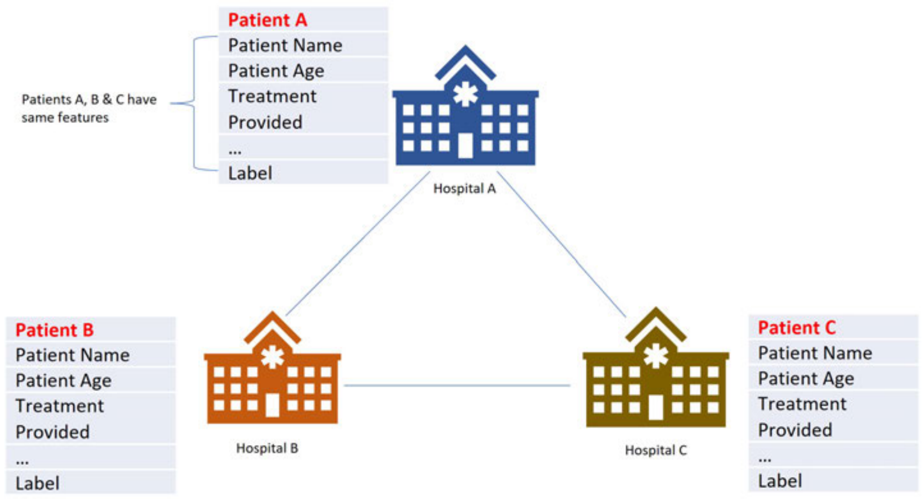


Figure 2.3: Horizontal Federated Learning

Vertical FL

With vertical federated learning, the data samples largely overlap among the users and differ in feature space. Hence, it also known as feature-based federated learning. The number of participants required for vertical federated learning is usually less than horizontal federated learning[186]. The datasets involved in vertical federated learning must be from the same sample space and a different feature space. The process includes collating different features such that the features are dissimilar whereas the data overlaps to greater extent.

The steps involved in vertical federated learning are as follows:

- Encrypted training samples are aligned and used for training as an encrypted model.

- An aggregator Z sends the key to the training parties X and Y for data encryption so that only encrypted data is transmitted over the network.
- During model training devices X and B can perform the intermediate gradient step and exchange loss values and then re-calculate the gradients after the intermediate gradient exchange.
- Newly calculated gradients are then sent to aggregator Z,

During the whole process of training and model parameter sharing, participants X and Y do not know about each others model features and the only part of model whose parameters are supported by the local model of X and Y are shared to them after global model aggregation. Vertical federated learning can be used by large organizations, such as the financial and retail sector as they can receive suggestions to obtain information on customer habits for a better business plan.

2.4.3 Federated Transfer Learning

To leverage the information gathered from other source domains, the transfer learning approach is widely adopted. Transfer learning is generally used to transfer the trained models to act as a base model which does not require training from the start. In other words, transfer learning allows the processes, training distribution and testing distribution to be different. Pre-trained models which are trained on generalized data sets are used as the base to start the training process. This can help the model become more specialized after training. Federated transfer learning is a special case which is different from horizontal and vertical federated learning.. The most important part of federated transfer learning is to identity which generalized model to use for particular training based on the training domain. Then, the chosen generalized model is trained to obtain an application-specific model that is based on the target domain.

2.4.4 Frameworks for federated learning

- TensorFlow Federated (TFF): TFF [63] is a platform on which federated learning can be performed using TensorFlow modules. It consists of two layers namely: 1) the federated learning layer; and 2) the federated Core. The federated learning layer provides high level interfaces so that existing keras and non-keras models can be integrated with TFF. This layer does not require the user to understand the underlying working and can easily use pre-defined functions to test the federated environment.
- Flower: Flower [9] is a novel framework for federated learning which supports federated training. It offers a stable and flexible ML framework for federated learning training. Similar to TFF, it provides high-level abstractions that allow users to use the pre-defined modules as is and can add their own modules on top of the existing stack so that the new resultant model is more job specific. For wireless devices, where system

heterogeneity prevails, system resources such as memory, bandwidth, and computational capabilities are not bottlenecks of the system.

- Sherpa.AI: This is a privacy-preserving framework for federated learning. It aims to bridge the gap between applied and fundamental research. Sherpa. AI [123] supports keras models and many models from scikit-learn including linear regression, logistic regression clustering algorithms.

- PySyft: PySyft [224] is an extension of PyTorch for secure and privacy-preserving deep learning for federated learning. It enables the use of 1) federated learning; 2) multi-party computation; and 3) differential privacy. It helps in communication and data transfer for virtual workers that are created for federated training. It implements a protocol for communication between a master node and network virtual nodes. It provides a tensor chain dividing a main tensor. This tensor requires two subclasses of SyftTensor class, namely 1) LocalTensor and 2) PointTensor. These tensors are used for the remote execution of operations on virtual workers and only updated results are transmitted back to the main central master node for aggregation.

- FATE : Federated AI Technology Enabler https://fate.fedai.org/(FATE) is an open source framework initiated by WeBank. It implements protocols based on multi-party computation and homomorphic encryption. It supports many federated learning algorithms including horizontal federated learning, vertical federated learning and federated transfer learning. A very important component in this framework is FATE flow which is responsible for a worker's job scheduling and their management. Generally, the following steps are involved in FATE flow for federated learning:1) Calling interface by FATE to upload the training dataset; 2)the pipeline for training is defined and FATE interfaces are called for job upload; 3)the training parameters are continuously adjusted as per the training results; 4)the dataset for prediction is then uploaded through FATE; 5)the prediction job is defined and then executed through FATE.

- OpenFL: Open Federated Learning [121] is a platform for federated learning developed by Intel Labs & University of Pennsylvania for healthcare related federated training. Regardless of its initial use case for the healthcare sector, it has now become agnostic to other use cases. Python is mainly used to code the framework and the distribution is done with the help of docker, pip and conda packages. The main idea behind OpenFL is to train the model on the remote workers with their private data. Data remains private to the data owner, however, only the updated gradients are sent back to the server for model aggregation.

- NVIDIA Clara: This is the application framework for federated learning which allows researchers to work on different strategies for federated learning. NVIDIA Clara integrates with itself NVIDIA runtime environment known as NVFlare which helps in getting the domain specific implementations of modules required in this framework: https://developer.download.nvidia.com/CLARA/Federated-Learning-Training-for-Healthcare-Using-NVIDIA-Clara.pdfNVIDIA CLARA

- FL4J: Federated learning for the JVM platform [64], uses Java as the programming language which allows code re-usability and allows training on android devices with no further code changes. To provide federated modules abstract model descriptions and algorithms are proposed.

- FLEE: Federated Learning and Early Exit of Interference [210] is a hierarchical framework for federated learning which provides distributed deep learning training over the main cloud, intermediate edge and local end devices. Using the architecture specified in FLEE, the network can be updated dynamically with no model resetting. FLEE consists of three parts which together forms the whole framework. They are 1) process specific to FLEE; 2) algorithms; and 3) mathematical formulas. In FLEE, the main large model comprises two part: the front and back part of the model. The first part of the model is used for deployment on end devices and the second part of the model is used for deployment on edge devices. The training process involved in FLEE includes the following steps: 1)Local updates: Using the local data available, the end devices train and update the network of model 1 (front part of the model); 2)Aggregation at edge: Since the process is at the edge level the edge devices are regarded as the servers for aggregation and the edge devices are the clients performing local training. Therefore, the local model parameters updated by the end devices on the front model (model 1) are then passed over to the edge server for aggregation at the edge level; 3)Update at edge and end device local update: Based on the model parameters obtained by the end devices, the edge servers then performs the entire model training (now including the back model) and sends the obtained results to the cloud server; 4) Final Aggregation at Cloud: In this step, the aggregation occurs at the cloud-level based on the parameters shared from the edge level training. Here, the model obtained is the global collaborative final model.

- FedLab : FebLab [198] is a federated learning framework with the option of customizable interfaces and high level-APIs for federated training simulation. The paradigms supported by FedLab include standalone, hierarchical, and cross-machine. FedLab consists of two main participants - The server and client with each having two modules, namely NetworkManager and ParameterServerHandler. The first module, Networkmanager, manages the messages between client and server whereas ParameterServerHandler performs algorithm optimization and is independent of NetworkManager.

- Flute: Federated Learning Utilities and Tools for Experimentation (FLUTE) is an open-source platform for federated learning simulation particularly offline simulation with an optional integration with the AzureML workspace.

The local workflow for FLUTE is given as follows: 1) the initial model is sent to the clients; 2) the global model instances are trained with the local data available on each client device; 3) after local training at the client devices is complete, the trained model parameters are then shared with the global server; 4) once the information is received by the server, it forms a new global model by accumulating all the received gradients or locally updated parameters; 5)further, there can be an additional or optional feature of having the training performed by the server before final aggregation.

Edge Federated Learning Consideration of edge federated learning [169]. Edge federated learning is a type of ML process that integrates both edge computing and federated learning to best utilize the data generated at edge devices while preserving the data privacy as data is not shared. EFL is a promising solution for preserving privacy and obtaining the collaborative predictive models. With edge federated learning, communication with the central server is reduced but communication between the edge servers is required which helps the intermediate communications and model aggregations at each level.

Chapter 3

Literature Review

Overview

In this chapter, we summarize the results of the systematic literature review. The goal of this chapter is to review the prior literature on stragglers and their management. Section 3.1 outlines the existing latest literature on stragglers in edge federated learning Section 3.2. summarizes the techniques in the literature, to mitigate stragglers and the motivation for the proposed research work.

Stragglers in the Literature: In this section, we review various studies in the literature that have attempted to address the straggling-effect. Numerous methods have ben proposed for straggler mitigation. The problem of stragglers has been studied by many research works in the literature as a distributed optimization problem.

Generally, for parallel distributed deep learning, parameter servers are more prevalent with master-slave architecture and the method of communication is synchronous. Traditionally, in parallel distributing learning, straggling-effect is a common problem. The work in [16] proposed the concept of the back-up worker to mitigate stragglers. The main concept was to increase the training speed by introducing redundancy in terms of additional participating clients. In this approach, a total of (N + b) clients are considered where b indicates extra clients for redundancy. During the training process, whenever, the first N clients respond, the server stops waiting for the other updates and performs global aggregation while discarding the updates received later. The work in [23] focus on exploiting stragglers by fixing the computation and communication deadline so that all the workers complete the training in the same amount of time and communicate with the server without any wait time. This reduces resource wastage, as stated in the earlier approach. Additionally, the former approach works well only when the number of stragglers is less than or equal to the number of back-up workers, b. If the number of stragglers is more than b, this resembles the ideal stragglers situation where the wait time is introduced.

Moreover, both of them follow a synchronous distributed training approach and are the approaches followed in distributed systems. However, these are also applicable to the context of FL where distributed training is performed without data sharing. Some studies such as [150] tried to avoid the straggler problem. The work in [184] proposes adjusting the training batch according to the computation time taken by the clients, where both static and dynamic stragglers are identified. However, the adjustment depends on the computation time of the workers at the previous iteration which takes more time to decide the mini batch size and update it in subsequent rounds. In [212], the training speed of each worker is managed by a parallelism manager. If stragglers are detected, by comparing the remaining time of a worker with the standard epoch time, its task is transferred to the fast clients to speed up the training process. This is similar to the work in [10], where data duplication among clients is seen as a way to mitigate the straggler problem in distributed systems. The algorithms for straggler mitigation in distributed systems mainly involve data duplication among clients, task re-assignment, and ignoring or dropping stragglers (as in case of federated averaging). These methods are not applicable in the context of FL since data is not considered a central entity at the cloud server. Hence, the problem of stragglers in FL is unique with its own characteristics and can result in a degradation of performance in the case of federated edge computing context since federated learning in edge computing can cause various issues such as computation and communication delays due to straggling nodes at the edge levels[117]. Therefore, the intermediate edge level communications remains halted thereby delaying the cloud communication at the next level. In another study [179], in order to deal with stragglers in the edge computing scenario, a large number of clients are selected by the edge servers so that even if there are straggling nodes, it will not have a major impact.

To alleviate stragglers in FL, which could be thought of as a distributed optimization without data centralization and dissemination among clients, various techniques have been implemented. For instance, ESync [84] was recently proposed which involves dealing with the blocking time of pioneers instead of focusing on the long-tail (straggling) effect. ESync tries to utilize the idle time of pioneers by allowing them to perform additional training rounds only if it can be accommodated within the predicted straggler response time. Based on the straggler response time, the state server decides if the training round of the pioneer can proceed or if it is allowed to synchronize. It is worth mentioning that if pioneers are allowed to perform too much local training, during the wait time for stragglers, it could diverge from the local minimum thereby adding inaccurate results to the global model. A different approach from ESync is introduced in TiFL [13]. TiFL selects clients with a similar response time into a same tier so that whenever clients are randomly selected from a tier, they take same time to return the updates to the server. Unlike the aforementioned methods, FedProx [81] introduces a proximal term to deal with system and statistical heterogeneity which are the causes of the straggler effect. The notion behind the addition of proximal term to FedAvg is reparameterizing it such that the local updates are not diverged from the global objective, thereby involving partial work of the clients by tuning it. While others focus on profiling clients into tiers(TiFL), restricting clients to diverge from the global objective (FedProx), utilizing the block time for pioneers (ESync), FedCS [102] focuses on client selection as an important step for federated training in mobile edge computing. The selection ensures that only clients who can perform model training and update it within the set deadline are selected. In this way, more clients can be incorporated in the training process to achieve high performance results unlike the baseline FL protocol with random selection restricting the number of clients per round.

In [85], the problem of stragglers affecting the over-the-air FL model aggregation is overcome by exploiting the benefits of relays. Moreover, devices with the help of relays can become communication efficient by uploading their results to the access point following the amplify and forward strategy of relays. In [202], the FL model is made efficient by estimating the contribution of participating clients by the iteration detection algorithm, thereby reducing the training time for FL. Further, the work in in [122] starts with faster clients and later allows slower clients in model training. Hence, near to target accuracy is achieved via the participation of faster clients followed by some information incorporated from stragglers towards the end of model training. There are various studies in the literature as in [85], [105], [221], [215] that investigate the edge federated learning scenario. FedEN is closely related to FedProx [81], as the latter also work on the client drift problem. Similar to FedProx, $FedEN$ tries to optimize the local objective such that the sparse models are obtained and at the same time parameters are shrunk to be as close as possible to the local optima. The work in [59] contain some of the early work on federated learning dealing with client gradient divergence and proposes the SCAFFOLD algorithm. SCAFFOLD focuses on client drift due to gradient divergence and proposes to add a correction term in the local objectives of edge clients which can nullify the effect of divergence and brings gradients as close as possible to the global minima.

Stragglers in Distributed learning: In [48], authors introduced a method named EP4DDL, which uses the technique of load balancing to over- come the straggler problem in distributed deep learning. The method proposed in the research uses the elastic parallel strategy to obtain load balancing, considering the real-time input from CPU or time per epoch or parallelism. Following this, the input is normalized and model prediction takes place. Finally, the output is then un-normalized. The result obtained is then used to calculate the participating node speed. In a nutshell, the delay time associated with stragglers is reduced by adjusting the iteration speed of each participating device. This is similar to the work in [212], where the iteration speed is adjusted to speed up the overall training process. Another approach in [47] is Berrut Approximated Coded Computing (BACC). Coded computing has a long history in the literature in dealing with stragglers. In coded computing, raw data is converted into coded data and then the coded data is used for all computations. Coding, in general, adds redundancy in data so the data is replicated among multiple servers. Finally, the results from the subset of servers can be used by the main server thereby reducing the time delay associated with stragglers. In relation to the current problem with coded computing such as computation instability, new approach such as BACC recently introduced, BACC allows the master server to share the coded data generated with the worker nodes and waits for the results from the fastest worker nodes and computes the final results. Hence, master server is reducing the overall training time. However, the results from worker node who might have valuable information might be missed in this scenario. Further, accuracy might not be as high as it would be when all worker node results are added together. Similarly, gradient coding methods are used for optimization in ML and are also used to straggler mitigation. In gradient coding, un-coded data is shared between the participating workers in such a way that it is replicated throughout the workers in the training process. Redundancy across all workers is achieved either through data creation and sharing or via data sharing at the initial stage of the training process. However, data sharing can't be done in federated learning as the sole purpose of federated learning is to provide privacy-preserving ML. The work in [11] adopts gradient coding to mitigate stragglers via a dynamic code word, GC-DC, assigned to workers for large-scale distributed learning. At iteration, the workers are assigned to various computing clusters to improve training efficiency by working on extra data but not by increasing the computational burden on workers. This helps in assigning the stragglers to different clusters uniformly. Another approach to deal with stragglers was recently introduced in [203] where only the parity blocks are cached over with write once and read multiple times technique for the workloads. However, whether it is caching or coding, data is pre-generated and then distributed among the participating nodes. This might speed up the training process but there is no guarantee that the pre-generated data improves accuracy. Apart from defining what stragglers are and how they manifest, the work in [135] defines a recovery threshold where the recovery of the system with stragglers is only possible when the number of stragglers is less than the number of actual devices in the training process. The authors propose a scheme to perform distributed matrix multiplication by adopting frame quantization providing a better reconstruction performance. Frame quantization is translated in matrix multiplication settings where more robust and efficient communication can take place with a varying number of devices in training process. In order to stop stragglers being dropped out from the training and to improve accuracy, authors in [94] proposes gradient coding with dynamic grouping where stragglers are distributed equally in each group and coded data is then distributed among all the devices per group. This ensures that the data is duplicated among all devices to compensate for straggler's delay. The main essence is that the devices are group together using a clustering algorithm, depending on how similar the gradient cosine is. While the aforementioned techniques deals with stragglers mitigate stragglers in different ways, the work in [8] detects if a device will tentatively become a straggler based on its performance since the early detection of stragglers helps in identifying them and also helps in their careful speculation, based on the training performance, in the heterogeneous environment.

Task speculation is done via Hadoop which records the progress of each task over time and compares it with a threshold called the average progress score. If a score is less than the threshold value then the task is considered to be a straggler. The algorithm proposed in [8] is a better way to identify a straggler as identification is the first step before mitigation. Identifying stragglers also helps to analyze the characteristics of stragglers which better helps to devise an algorithm for dealing them. The recent work in [219] considers distributed code computing to deal with stragglers. This approach is similar to the work in [47], [11], [203] where data replication is of high interest. However, the coding approaches add additional similar data for processing which seems to be better in ensuring the system is fault tolerant. However, accuracy cannot be improved as it depends on the new and valuable information incorporated from the devices, not on the replicated data. Replicating tasks to improve training efficiency is inefficient as the overhead on the fast devices increases due to additional task processing. Different from the work in [203], the authors of [134] do not take into account parity blocks, instead instances of cloud-hosted models are deployed by workers. In addition to this, because the hosted model is independent of encoding and decoding procedures, any neural network architectures can be used for the training process. Compared to the traditional techniques used for dealing with stragglers, ApproxIFER is agnostic to the model data and architectures and can deal with any number of stragglers in the system. Authors in [115] tries to minimize the tasks completion time by considering trade-off between diversity and parallelism. Efficient federated learning is proposed in [188] where FastSlowMo is introduced. FastSlowMo is combined with Nesterov Accelerated Gradient (NAG) and client momentum. The steps followed in FastSlowMo are as follows: 1)Each participating device performs local training, update their local model and momentum after each epoch with the help of NAG 2) server aggregated the momentum from all clients and returns the average value back to them for further training rounds 3) server also receive and aggregate local models after each training round.4) updated model is broadcasted again to all the participating clients. The main benefit of this approach is that the momentum of both server and clients are used to accelerate the convergence performance as compared to approaches with momentum only at the server side. The download communication cost of the participating user is assisted with the help of servers in [74]. The main benefit of doing so is reducing the effort required for downlink communication required by the resource constraint participating users. In other words, communication cost is given to the helper servers to support the users thereby reducing the overhead on them. As compared to all the techniques used for straggler mitigation in literature, studies in [6] is the unique approach which jointly works on the replication of tasks as well as it's speculation and load balancing. However, none of the three approaches mentioned are best among others as their response depends on the load in a network as well as on the variability of the tasks coming in. Generally, when the load conditions are less, then the replication of tasks can give a better response times. Similarly, when the load in the network is moderate then speculation can be of great help in order to identify any deadlock or an overhead in the network. In another study [152], performance of the participating nodes are monitored and the failing nodes, which are stragglers, are identified. Once stragglers are identified then they are not allowed to participate thereby avoiding them totally once identified.

Synchronous methods are where the parameter server waits for all the devices to return with their gradient updates which increases the wall-clock time, causing straggler emergence. Hence, to deal with stragglers causing delay in the training process and to allow faster synchronization, asynchronous methods are adopted in the literature. With asynchronous methods, the parameter server does not wait for any of the participating devices before proceeding to the next round of training. Instead, as soon as the updates from the devices are received, the gradients are updated. Hence, synchronization speed increases but sometimes at the expense of accuracy as stale gradients can cause the divergence of gradients to a greater extent if the stale gradients are calculated at the older version of the model [222], [89], [22]. Synchronous methods are better in terms of accuracy but incur a higher wall-clock time, whereas, asynchronous methods are highly time efficient but can result in a trade-off with accuracy. Therefore, for asynchronous methods, stale gradients calculated on outdated parameters beyond a certain threshold can neither improve accuracy nor allow the algorithm to reach convergence due to gradient divergence. The use of gradient staleness with an upper bound is suggested in [5] where it is discussed that there are certain parameters such as the number of workers, and the batch size which can help in adjusting the gradient staleness. Hence, gradient divergence can be reduced thereby improving the model's prediction accuracy. Different from the aforementioned methods, online min-max optimization is proposed in [167] where nodes with high cost are considered to be stragglers and since the algorithm is gradient and projection free it is considered to be computationally efficient. The algorithm is proposed for online training and is called the Distributed Online resource Re-Allocation (DORA). In DORA, after the careful speculation of stragglers performance in online training, the resources are meticulously distributed among them to help improve the training performance. Similar to [47], [11], [203] which used gradient coding for straggler mitigation, the authors of [7] proposes an erasure coding scheme where instead of a single server, multiple servers process some k version of coded sub-tasks such that if the fastest k servers process all the sub-tasks then the whole task is complete. Further, the decoding time required to decode a sub-task is negligible compared to the processing time. This type of method creates redundancy in data to increase the processing time in the presence of stragglers in training process. The following considerations make [47], [11] and [7] different from other studies: 1)the general number of coded tasks are provided initially; 2)the start time of a sub-task is generally random and it varies with respect to the completion time of the sub-tasks; and 3)there is be a trade-off of time between the server and task completion time. To improve the quality of service of a system [156] proposes Straggler Prediction and Mitigation (START) which not only mitigates stragglers similar to other algorithms, it also identifies stragglers at an early stage. In other words, START predicts the devices which can become stragglers later thereby reducing the time and cost in identifying and mitigating stragglers. This is executed with the help of LSTM (long short-term memory) based neural networks which can record changes in performance in terms of CPU, amount of bandwidth consumed, service level agreements and many more. Hence, the changes in the dynamic cloud-like environments can be recorded and a prediction can be made as to whether a device could become straggler in the next cycle. Similar to [48], [15] devised worker coordination technique based on load balancing. Two worker coordination schemes are used in literature: asynchronous parallel (ASP) and stale synchronous parallel (SSP). ASP is where the server doesn't wait for stragglers and update the global parameters as soon as it receives updates from the fast workers. This improves training efficiency but due to the incorporation of stale gradients, accuracy decreases due to gradient divergence, whereas in SSP, the wait for stragglers is only until the threshold which imposes a bound on the gradient staleness. Therefore, SSP is a type between ASP ans BSP. In [15], the basic idea is that the server issues multiple copies of straggling tasks and collects only the ones which reply first. Load balancing can be done in either a with static or dynamic way. Static load balancing, does not require any real-time progress whereas in dynamic load balancing, straggler detection and load migration is done for specific groups at a very fine granularity level and in real time. An important assumption made for dynamic load balancing is that the sequential process of the jobs is not computationally efficient. Hence, all the samples in a batch are processed as a single tensor matrix in dynamic load balancing for model ML workloads. Therefore, in [15], the following properties were investigated when the proposed algorithm was developed: 1) How practical is the algorithm? 2) How effective is the algorithm with current ML frameworks; 3) How efficient is the algorithm in terms of not interrupting the processing time. The work in [83] investigates the coarse-grained and fine-grained performance of the models used for distributed SGD. A heuristic based loss function for coarse-grained models produces the results similar to fine-grained models and were evaluated per day. Then, the asynchronous model is chosen for the framework which increases the average prediction accuracy or the test errors. A fine grained-model can be an abstraction which simulates stochastic gradient descent for many workers with a lower transmission rate. The architecture used is the parameter-server so that resources are shared among multiple workers. Models were evaluated under the following three circumstances as follows: 1) homogeneous workers with an overlap between communication and computation 2)homogeneous worker without the overlap between communication and computation; and 3) heterogeneous nodes are included in the model formation with coarse-grained model parameters obtained easily. Hence, the model was thoroughly tested in different scenarios for architectures including centralized and decentralized settings to ensure it was architecture agnostic.

A decentralized framework, ScaleReactor, is proposed in [218] to mitigating stragglers . ScaleReactor decouples performance from the scheduler. It monitors, detects and identifies the straggling resources. Therefore, the performance interferences are migrated to container based clusters thereby reducing performance loss. Hence, the computation and time complexity as well as offline profiling approaches is handled by ScaleReactor for better resource utilization.

The authors of [106] proposed a preemptive online algorithm which determines the distribution of tasks between the participating workers. In addition to this, the overhead associated with a preemptive algorithm is taken care of with the help of a non-preemptive scheduling algorithm. In order to improve the performance of the graph convolutional network, which is affected by straggling nodes, Boundary Node Sampling (BNS) - a graph convolution network is proposed. This method partitions the graph which reduces the number of straggling nodes. Hence, both communication and memory associated costs are reduced. In other studies [19], the group based sampling technique, ReSKY, is proposed which helps in the processing of worker nodes considering the sub arrays of other worker nodes in an efficient manner. Hence, this decreases the straggling effect as well as the cost associated with network delay. The assumptions made for the straggler problem are as follows: 1) a database system has a single coordinator worker node and under that coordinator are multiple processing worker nodes; 2) all pieces of data including the overlapped data in the array are distributed among all the participating worker nodes; 3) all participating worker nodes will have an equal number of non-empty chunks of data. Following is the distributed processing approach in ReSKY: 1) each participating worker node computes the representative of its sub-array; 2) after forming the sub-array group, each worker node identifies its local sub array group skyline; 3) the coordinator then computes the global sub-array group skyline and this helps in pruning the sub groups. This third step in the algorithm is where the group bases sampling method is introduced to mitigate stragglers. The authors of [87] proposes a dependency and resource aware scheduling algorithm for jobs that are heterogeneous in nature and can cause bottleneck in the system thereby increasing the straggling effect. The proposed algorithm reduces the latency associated with stragglers and improves the utilization of resources effectively in the cloud environment. The main idea of Fregata is that it splits the task into various sub tasks by taking into account, the constraints of tasks. Finally, it assigns tasks which are independent of each other to different processing worker nodes such that all tasks can be processed in parallel, thereby reducing latency. To deal with the disk associated I/O overhead with random walks, an in-storage accelerator for random walks, known as FlashWalker, is proposed. In FlashWalker, if Clue Web has more walks then the processing straggler nodes takes a small portion of processing time. Hence, the optimization is effective to process the non-straggler processing nodes with improved performance. The hypergraph partitioning based Online Joint Scheduling (HOJS) technique is proposed in [136] where it deals with the impact of diversity in resources during the scheduling process. HOJS proposes a time associated cost based on the process completion time and proposes an optimization algorithm which can efficiently describe the scheduling of processes thereby reducing the overall completion time of processes. HOJS makes scheduling decisions, and the hypergraph dynamically gets updated based on the previous results. It can generate an efficient scheduling techniques since all the constraints such as resource diversity and distributions are considered at the time of scheduling. However, the current implementation the optimization of data replication layout is not considered.

The authors of [114] proposes a gossip-like protocol in IoT settings. The main idea is to analyze the dissemination and collection of the delays for transmission models by introducing redundant IoT relays and erasure coding techniques to deal with stragglers. Hence, the system performance is evaluated based on the covertness estimation as well as with message passing associated straggler delay. Further,it is near impossible for cohort estimation as well as message passing design parameters to be chosen at the same time simultaneously. Similarly, in [171], the authors specify some of the most popular techniques that are used to mitigate stragglers in the literature and proposed the Heterogeneous Throughput- Driven (HTD) scheduling algorithm. The main idea is to study the problem of batch- processing in the heterogeneous settings which can cause straggler problem due to workload overhead. For a single job execution, a task scheduling scheme for heterogeneous settings is estimated. Further, for multiple sets of job coming in distributed settings, an optimal execution sequence is generated using parameter estimates. Further, the generated sequence is used to optimize the details of the tasks in the heterogeneous environment thereby obtaining the final scheme for task execution. Obtaining the optimal sequence execution list ensures that stragglers are mitigated. The following steps are used to obtain the optimal sequence for any tasks in distributed systems: 1) the original set of jobs are simplified and merged with the corresponding jobs thereby reducing the number of jobs in the list. Hence, the obtained output is the simplified set of the job list; 2) the obtained set of jobs are further sorted to obtain the optimal sequence with a bounded value for the completion time of each job in the optimal sequence list; 3) all the sequences obtained are verified using the pruning scheme. The verification of a task is for all the sequences whose completion time in the list is less than the bounded value. Therefore, the algorithm helps in the adoption of reasonable job execution optimal sequence time and ensures that all jobs are processed in a short duration time thereby reducing the straggler effect.

Stragglers in Federated learning In recent studies, such as [13],[81], [202], [215], [122], [42], the point of focus was on stragglers in federated learning. The basic ideology is to deal with stragglers who slow down the training process by not responding to the server, thereby delaying the global model aggregation. Different algorithms have been proposed to deal with stragglers in the aforementioned studies. For instance, segregating stragglers as per their training speeds in previous rounds, making local functions smooth with l_2 regularization, and better client selection, to name a few. Compared to synchronous methods, asynchronous methods are much more preferable and have been widely adopted as they improve federated training efficiency by performing global aggregation as soon as they receive the fastest worker updates. Therefore, the issue of stale gradients is common in asynchronous federated learning. However keeping a threshold on how much staleness is allowed helps in negotiating the trade-off between accuracy and gradient staleness. The work in [108], proposes a gradient staleness based algorithm, namely Sageflow, to deal with stragglers in federated learning. They proposed a two-stage solution based on grouping and filtering where small a amount of public data is used at the server for training. Sageflow can deal with stragglers by grouping then in different groups. However, Sageflow works better when a small amount of public data is available at the server to perform grouping and filtering of stragglers with a non-straggler device. Coded computing which is widely used in distributed systems is integrated with federated learning [100] and termed as coded federated learning. Coded federated learning allows the server to perform extra redundant computations which are computationally efficient for participating edge devices. Edge devices compute partial gradients based on their parity data and submits partial gradients to the server. The task of server is to aggregate the partial gradients calculated using parity data from the edge devices since only a subset of some of the partial gradients can be used for full gradients. Different from CFL, [101] proposed a buffered algorithm, named FedBuff, which is an asynchronous algorithm with no penalization for mitigating stragglers. The main idea is to use trusted execution environments (TEE) for buffering or storing updates from the participating edge devices. The procedure used in FedBuff is as follows: 1) workers receive model parameters from the server and perform local federated training in an asynchronous manner; 2) when the results are shared in an asynchronous manner, the server does not update the global model parameters as it should in the case of asynchronous techniques. Instead, the updates from the devices are stored in the buffer and the server chooses to have a threshold for how often the model is to be updated. A study in [29], proposed a variant of federated averaging which has additional features to deal with stragglers and detailed a new algorithm called as 'memory-augmented impatient federated learning (MIFA), which has two additional features - the first being impatient with respect to stragglers in the training process which is similar to federated averaging as it does not wait for straggling devices and ignores stragglers; and the second which includes the augmentation of the latest updates with the stored updates and then finally performs averaging of the gradients to obtain the final results. Since MIFA ignores stragglers in a similar way to federated averaging, it follows the synchronous technique for model aggregation. Different to synchronous or asynchronous techniques, [205] proposed semi-synchronous technique for stragglers known as clustered semi-asynchronous federated learning (CSAFL). CSAFL is a combination of both synchronous and asynchronous techniques where gradient staleness is treated by dividing the workers of different objectives into different groups. Further, the delays associated with model training and results submission are limited by a threshold. Similar to [156], the authors of recent work [76] predict the training workload of each device based on the workload completion history of the workers so that stragglers can be identified and dealt with accordingly. The algorithm is named as FedSAE, which is a synchronous and adaptive algorithm. The main concept behind FedSAE can be explained as follows: 1) the server can delegate the workload among the participating devices based on their historical information in relation to how they performed during training process; 2) active learning [182] in device selection. In a nutshell, the very first step in FedSAE is device selection based on the historical performance while in FedAvg clients are selected randomly. Next, the server predicts the workload for all the selected devices to speed up the training process while in federated averaging, all devices have some amount of workload for each training round. While federated learning aggregates the model based on the sample size, FedSAE aggregates it on the basis of the weights from the training loss of the participating devices in each training round. FedSAE is a better approach as it deals with the fairness of the local models and obtains a an fair global model with better convergence properties. Compared to the benchmark algorithm, FedSAE achieves a 22.19% increase in test accuracy. The algorithm proposed in [109] deals with stragglers due to system heterogeneity and communication inefficiency. To mitigate stragglers, the authors propose a simple yet novel algorithm named the Adjusting Mini-batch and Local Epochs (AMBLE). AMBLE is an adaptive algorithm that adaptively adjusts the local batch size as well as the local epoch size for devices which are heterogeneous in terms of system characteristics. AMBLE adopts a synchronous mechanism to aggregate model updates after each iteration. Since the local batches and the epoch sizes are adjusted, each device has a different mini-batch size as well as a different local epoch size with an adaptive scaled learning rate. AMBLE follows a simple yet unique approach for straggler mitigation which stands out of all approaches in literature. However, the scalability of AMBLE with respect to the number of clients in the training process is not clear. For computational challenges of the devices in federated training, [177] proposed FedAdapt recently. FedAdapt implements three techniques to deal with stragglers. Firstly, it offloads the layers of the deep neural networks used in federated training at the servers. The deep neural network layer after which the partitioning takes place for offloading is the offloading point. Secondly, reinforcement learning is used to identify the offloading point for each device before the actual offloading at the server can be initiated. Lastly, reinforcement learning is optimized so that the bandwidth is optimal at the time of the offloading procedure. FedAdapt reduces the training time up to 40% compared to benchmark federated averaging. Since reinforcement learning is used for the offloading procedure for each device in federated training, any increase in the number of devices requires more reinforcement agents to perform the offloading strategies.

In [127], the stragglers in a cellular network are identified and mitigated. The authors designed a mixed-integer optimization problem where the first part of the optimization problem deals with scheduling devices with a large amount of data and the second part deals with making the appropriate channel conditions for those selected devices to make transmission easier and successful. To achieve the first step of the chosen optimization problem, the authors of this study performs device selection using a greedy algorithm which helps in selecting devices with better transmitting capabilities, and it is worth mentioning that as the number of devices in the selection increases the data for federated training also increases thereby causing the data to move closer to the actual distribution over the population. Secondly, the appropriate design of a receiver combiner is proposed to ensure the channel is optimal for the signals to be received by the base stations at the end of the training round for final signal aggregation.

In [35], training efficiency is improved by minimizing the training maximization problem for 6G to enable mobile edge computing networks and AceFL was proposed to accelerate 6G-enabled mobile edge computing. AceFL takes into account both system and network heterogeneity which causes edge devices to become stragglers. The authors investigated the edge federated learning framework. The main aim of AceFL is to improve training efficiency in the presence of resource-constrained edge devices. There are two parameters that AceFL tries to optimize in order to achieve it's aim. The optimization parameters are local inexactness and band allocation frequency for each of the device. By adjusting the aforementioned parameters, the training efficiency of AceFL is improved and the two aforementioned heterogeneities are dealt within edge federated learning. AceFL allows edge devices to have local inexactness in the local objectives and use the proximal term in the optimization of the local loss function of edge devices. The idea of improving training efficiency in the presence of stragglers in edge federated learning is a hot area of research of recent times. However, the concept of using local inexactness and proximal term in the local objectives of edge devices is same as that proposed in [81]. the authors of [93] propose a convolutional neural network and multi-layer perceptron based intrusion detection model known as FedBatch and to deal with stragglers in FedBatch, they proposed the federated aggregation algorithm where the average is calculated with respect to uploaded weights only. The asynchronous distributed federated learning proposed in [116] shows that the system can perform better even if it consists of stragglers and that the model convergence speed can be improved if the asynchronous technique is used for model training. In [17], the authors propose dynamic asynchronous and anti-poisoning federated learning (RAPFDL), an adaptive algorithm, where stragglers with characteristics such as low computation power, and improper channel conditions are removed from the training process. This is similar to the benchmark algorithm, federated averaging, where stragglers are completely ignored and any information from stragglers is not incorporated in the training process. System stragglers are identified in [67] and FedScale, a federated learning benchmark suite is proposed. FedScale provides high level APIs which can be used to develop federated algorithms with less effort. Further, FedScale provides realistic datasets to reflect the characteristics of federated learning. The authors of FedScale finds that the stragglers in the system can slow down practical federated learning to a very great extent. They provided the client_completion_handler(), an API for dealing with stragglers. Heterogeneous multi-view data is investigated in [43], giving rise to stragglers in federated learning. To mitigate stragglers, they developed an iterative optimization algorithm which helps in providing flexibility between communication and computation. The work in [82] proposed FEDDATE-CS to provide the deadline and decision for client scheduling in federated learning within a multi-armed bandit framework. To deal with stragglers, the authors proposed hard deadline for each local training round. Eventually, this hard deadline threshold mitigates stragglers thereby improving training efficiency. Both gradient staleness as well as the non-IID characteristic of data reduces the model utility. K-async federated learning has two issues: stale gradients causing gradient divergence and non-IID data generated at each device causes a huge impact on the model utility as the distribution does not matche with the overall distribution [168]. Weighted K-async federated learning (WKAFL) was proposed to deal with these two issues. WKAFL works in two stages and performs the following: 1) stale gradients are clipped to make model efficient and stable; 2) the model works perfectly with improved utility and robustness to non-IID data among edge devices. The authors of [185], imitated the heterogeneous system by keeping constant and dynamic local epoch steps in federated training. To perform a fixed number of local steps, each participating edge device performs some fixed local update steps for training. Similarly, for dynamic local steps, a random number of steps are performed during local training. Hence,local steps applied in conjunction with the learning rate plays an import role in the training process as it can improve training efficiency. This is similar to [109] where mini-batch size as well as the local epoch size plays a crucial role in training efficiency. In [187], the heterogeneities of IIoT devices are considered to be the source for straggler manifestation. The authors proposed a DT based IIoT architecture to retrieve the real-time statuses of IIoT devices.

Further, to improve training efficiency and to mitigate stragglers, an asynchronous algorithm is proposed. The proposed algorithm is based on Deep Reinforcement Learning (DRL) which is used for participating edge device selection and for clustering algorithms to classify devices based on the size of their data and computational power. The work in [36] proposed the Adp-FedProx algorithm in order to achieve optimal training performance. Adp-FedProx has better convergence than asynchronous federated learning and asynchronous federated learning has better convergence than FedProx as Adp-FedProx can dynamically adjust the number of global iterations. Hence, Adp-FedProx improves the training efficiency by 5% to 10% compared to asynchronous federated learning and FedProx. The authors of [20] proposed an in-network computation framework for device scheduling in edge networks to decentralize the aggregation process thereby optimizing communication and reducing the computation overhead on the edge devices. The core part of mitigating stragglers is the scheduling process which is based on a power law distribution. Further, the authors derive the theoretical bound on the performance of the participating edge devices. In [90] the asynchronous technique, Fed2A, is adopted only for the local stage. That is, the edge devices can load the parameters from the server, perform training with their local data and upload the updated model parameters back to the server for model aggregation. However, the critical part is at the server where it has to decide when to aggregate all the updated parameters. Hence, the server continuously receives the updated parameters from the asynchronous edge devices. After receiving the parameters until the maximum wait time of the server or when certain pre-defined conditions are met, the server starts the global round of model aggregation and the updated model is obtained. This updated model parameters will then be broadcasted back to the edge devices for the next round of training. Fed2A achieves higher accuracy and better training performance compared to the benchmarks FedAvg, FedProx and FedAsync. However, Fed2A only discusses the model parameters upload stage and the reverse step is not discussed, which is downstream optimization. Worker device management or scheduling plays a crucial role in efficient federated learning as it all starts from the edge device which will train a local model and any heterogeneous characteristics associated with the devices can cause them to become stragglers. In [138], the authors propose fairness at the device level which is with respect to model loss or accuracy loss for the participating edge devices in model training. An efficient and fairness goal oriented algorithm, Eiffel, is proposed to consider the resource constraint environment of the edge devices. Instead of including a random number of devices which could otherwise be resource-constrained and can stop the training process, Eiffel undertakes intelligent device selection and can save essential resources such as channel bandwidth, and reduce the energy consumption.

The authors in [120], deals with stragglers by utilizing a better client selection procedure for federated learning in a hierarchical framework. Stragglers are those devices whose wireless links are adverse and the devices have low computational capabilities. The main objective is the identification of stragglers in hierarchical federated learning. Based on the multi-armed bandit framework, a context-aware online client selection policy (COCS) is proposed. In COCS, the network operator select the clients on the basis of their computation and communication requirements and maximizes the resources required for local training assistance. Edge clients are usually charged for the amount of resources allocated by the network operator. This proposal is similar to [102] and [76] where the selection and scheduling of clients is of utmost priority. Despite client selection being the better approach to deal with stragglers, there are other reasons for straggler emergence in a network which is pointed out in [40]. The authors of [197], provide a detailed overview of some of the approaches to deal with stragglers in federated learning. In addition to all the aforementioned approaches which are used to mitigate stragglers, the authors of [141], propose an asynchronous algorithm which ignores stragglers and reduces the training time compared to other approaches in the literature. Ignoring stragglers (not involving them in the training process) is also the assumption made by benchmark federated averaging. The partial model aggregation approach, FedPA, is investigated in [88].In FedPA, the models for aggregation are known to be the aggregation number for a particular round which is identified using reinforcement learning dynamically. The goal of FedPA is to utilize valuable information from the stale models without discarding them. This is achievable only when the weighting factors of stale models are used along with the updated fresh model in the aggregation. The authors of [69] adopts an asynchronous approach in wireless network settings to deal with stragglers. The main goal is to adopt the asynchronous approach to obtain some amount of learning which otherwise is affected by the stragglers in the training process. In addition to this, limiting the round time of stragglers or uploading the fraction time in each training round is investigated in [97]. In other studies such as [68], to trade off between the statistical and system efficiency, the statistical utility of the client is associated with the utility of the global model. That is, the updates from the first K clients are collected out of a total $1.3K$ clients, with K being 100 by default), in each training round. Therefore, the time-to-accuracy performance is improved by selecting clients with high system and statistical utility. To deal with federated client heterogeneity in mobile edge computing framework, the authors of [172] introduced FedCH. Similar to other approaches in the literature, FedCH adopts hierarchical model aggregation. But, different from other approaches, FedCH takes into account efficient cluster topology where clients are formed in K clusters depending on their training capabilities. Therefore, each cluster will perform federated training independently and have their own training speed. And, the clusters where the participating clients are stragglers will not halt the system performance as they are restricted to a cluster and are not an important part of the system which is different to the work in [102], [82], [120] where client scheduling is the approach adopted. However, [159] not only deals with client scheduling but also resource blocks allocated to the clients depending on the communication and computation limitations for the participating clients. The authors of [220] deal with the communication latency associated with network stragglers using a distributed optimization algorithm, Delayed Gradient Aggregation (DGA). The main objective is to delay the aggregation process so that the communication can be delayed for the computation wait in the future training rounds.

In the work in [1], a reduction in the number of communication rounds between the server and the edge devices is considered. The authors proposed a hierarchical federated learning framework to address the challenges such as resource limitations associated with the edge devices as well as the data-privacy issue associated when performing model training. The main idea is to formulate the optimization problem for the scheduling of edge devices. In addition to this, resources are also allocated to the participating edge users based on the distribution of data. However, tight synchronization between the federated training workers is not considered. The authors in [217], investigate asynchronous federated learning. For each training round, fixed interval is provided with aggregation done in first-come-first-server approach. In addition to this, the number of clients selected in the very first round of training is N followed by only one client being selected for the subsequent training rounds. Therefore, with the help of model diffusion approach, clients participating in the training rounds can be organized for each training round. Further, the staleness associated with the asynchronous approach is dealt with the adoption of the weighted average aggregation. Similarly, the work in [130], investigates bandwidth allocation and the client scheduling problem, which can cause stragglers. Further, convergence bounds guarantee that the trade-off between the latency associated with every round and the total number of training rounds helps to achieve the required fixed accuracy. Similar to our approach, [125] investigates the percentage of stragglers present ranging from 0- 90% in the training process. The authors propose different federated learning system parameters such as the number of participating clients, the number of stragglers in the system, and the communication cost. The consideration of straggler percentage in the system is similar to the work in [81] where system heterogeneity was accounted for by incorporating the straggler's presence percentage in the system. Including a large number of clients can speed up the training process only when the percentage of stragglers in the system is less compared to the number of non-stragglers. The authors of [14], investigate the limitation on the number of devices participating in the training process which reduces the metrics such as communication cost or carbon emissions. However, the issue of generalizing the train samples when a large number of clients participate in the training process, the effect on generalization is not studied. Similarly, the work in [32] deals with resource management in the wireless network given the problem of client scheduling as well as energy management among the participating devices. To address the challenges of statistical and system heterogeneity, few modification to benchmark algorithm can be done and accuracy over communication rounds can be used as a metric for federated training which was investigated by authors in [223] who proposed EasyFL. EasyFL provides diverse training methods as well as scalable deployment with the assistance of containerization and service discovery.

The interface layer and system layer forms a two-tier architecture. The interface layer provides all the APIs required for the system which makes it complexity hidden from the end user, whereas, the system layer consists of eight modules that provides additional functionality to the system. In [95], all the components contributing to the delay in the training time, such as the overall computation time, bandwidth link delays as well as the straggling effect, are considered. They improved the system training speed and system throughput by proposing a system design topology based on the max-plus theory of linear systems. Fault mitigation is similar to mitigating stragglers in edge computing as investigated in [31] and they proposed the FedFM (Federated Fault Mitigation Algorithm). In FedFM, experiments are conducted to analyse the effect of increasing edge devices that fail due to network issues. Therefore, the authors introduced a network architecture where the participating edge devices and the server never changes their topology. Hence, they are static in nature. Further, nodes can participate and can opt out dynamically. However, the concept of devices joining and opting out dynamically can adversely affect system efficiency as well as accuracy. The authors of [216] specify the synchronization approaches and their subsequent results. For stragglers, decentralized algorithms are the best option as they have better solutions for mitigating stragglers compared to synchronous approaches which can get better convergence results. In the enterprise federated learning platform [99], the authors consider that the participating devices are moderate enterprises while the data they hold is large. The concept of the proposed algorithm is to introduce a timeout parameter. This timeout parameter allows the server to wait for a certain amount of time, which enables the participating organizations to train and update their local results. Hence, once the timeout timer expires then the aggregating server doesn't wait for the model update results from stragglers and finally aggregate and form the global model. The concept of using a timer is generally used in distributed systems in the literature. The timer gives a threshold allocated as to how much duration of time a parameter server has to wait for the rest of the participating devices before proceeding with the model aggregation for a given training round. The authors of [69], proposed an adaptive algorithm for deadline determination. The deadline is adaptively chosen depending on the performance of the participating edge devices. The following steps are involved in the proposed algorithm: 1) Based on the collection information of computing resources, channel capabilities, the server calculates the transmission time and other important system parameters for the participating devices; 2) Based on the calculated transmission time and other parameters, the server provides deadline to exclude stragglers from the training rounds. Therefore, server does not wait for the straggling device to delay the aggregation process by fixing a deadline onto the straggling devices local training for each training round. Fixing a deadline for training rounds is a concept widely and generally adopted in the literature in distributed learning. However, ideally deadlines for training rounds alone are not sufficient for straggler mitigation. In [77], a training delay is identified over fading channels as well as due to wireless connection issues. Participants are allowed to perform one local update per global iteration round. Further, saddle point approximation is introduced to deal with the delay distribution in the wireless setting to ultimately make federated training efficient. A different approach to straggler mitigation is adopted in [75] where the stragglers are considered as the optical network units with limited bandwidth resources. The main idea is to reserve bandwidth for the participating units which can make federated learning efficient. Further, when the straggling units opt-in or opt-out, the reserved bandwidth can be adjusted dynamically which increases the utilization of resources, whereas the authors of [39], proposes a mechanism which can help to improve training efficiency which otherwise is affected by larger models. They introduced importance based pruning which helps in the extraction of a sub network taken out of the original network as its footprint. In another study, a tri-layer federated learning scheme is proposed to deal with stragglers. The three levels are defined as follows: 1) sample-based pruning; 2) resource and activity aware client scheduling process; and 3) local objective generalization for participating clients. In sample based pruning, model size is reduced which will eventually reduces the computation overhead on the training clients. The pruned layers are the ones with the smallest absolute layer-wise value which can have the least effect on its pruning. For pruning 1) the server collects sample data from the environment. That is, the clients are asked to provide a small amount of data they would like to share; 2) global model initialized with the received data from clients and the model parameters are shared with clients. Further, global model is updated in the subsequent rounds after receiving new updates from all clients; 3) sample based pruning is performed until the target pruned model is achieved; 4) the pruned model is shared to the participating clients for federated training; 5) in the case of a shortage of resources, the participating clients are allowed to submit their partial work to the server. The server aggregates the received locally trained models to form a global model. For the resource & activity aware scheduling process, the server sends or pushes the task with the minimum required system requirements. The available clients responds back with the resources available. These resources could be bandwidth, RAM, or data capacity for training. The clients with the appropriate system resources are selected for a given task. Finally, once the clients are selected, training is performed, and the results submission depends on whether the participating client is submitting a complete result or partial results. The clients are allowed to submit their half-way training results to provide their contribution. Therefore, stragglers are not dropped out or ignored. Instead, they are allowed to partially participate in the training process to contribute to the training process. However, the partial results submission can cause a decrease in accuracy as it causes gradient divergence. System heterogeneity in federated learning is an inevitable source of stragglers. With the consideration of system heterogeneity in [2], the authors proposed quantization as a possible solution which is a promising way of straggler mitigation. The main idea is that the server makes per-client decisions and selects the version of the model which is suitable for a clients and sends them the selected model version. However, the server maintains the model in the floating-point format and does model quantization based on the client's capabilities at each round during the configuration phase. Further, the quantized deep neural networks are used for federated training which has the flexibility of adjusting the different bit modes depending on the requirement. Similar to the other client selection and scheduling mechanisms mentioned before in above literature, the authors of [196] propose a joint mechanism for client selection as well as the effective management of resources. The process used to select appropriate clients is as follows: 1) all edge clients are sorted in ascending order based on the values of η_i as mentioned in [196]; 2) if more clients are selected then it can eventually increase the wait time for model aggregation thereby causing delay in model convergence. Hence, each client-to-be is evaluated with the existing clients to check if they align with the characteristics of the existing clients which will not have an impact on training efficiency and can still meet the training deadline. If a client can meet the deadline, then it is included in the training process else it is excluded from the selection; 3) this process continues until the appropriate number of clients are selected for the federated training.

Most recent works: In this section, we present some of the most recent literature on stragglers in edge federated learning. In [34], the emergence of stragglers due to communication resource limitations is considered. To deal with these type of stragglers, fixed step size adaptive fastest k SGD is introduced. Therefore, in the fastest k-updates algorithm, once the global model fastest-k updates are obtained, the updated model is broadcasted to the participating clients. Following this, the client calculates the partial gradients of their local data and the previous updates from these clients are discarded. This, therefore, improves the quality of the full gradient. In the adaptive fastest k-updates algorithm, the number of clients for which the server waits increases over time. However, if the number of stragglers increases then the wait time for the server can delay the aggregation process. In addition to this, the work in [12] introduces a straggler-resilient approach. The main idea is to combine a stale synchronous parallel parameter server (stale gradients are allowed in model aggregation) with a back-up worker(some k-backup worker are considered which can utilized later when stragglers increases) scheme. The advantages of using back up workers is that they allow the stale models to be used in global aggregation. This will ensure that all worker are busy and no one is idle after completion of their tasks and waiting for others. In [158], authors measure the system robustness with respect to the probability of stragglers being present which can hamper the training performance as well as accuracy. The results obtained shows that there is only a 2% drop in the algorithm's performance compared to the benchmark algorithm. But, the results achieved are with respect to all the clients participating in the training rounds. However, the problem of stragglers is not completely solved in the study since the increase in the number of rounds increases the client's model overwriting, thereby decreasing the accuracy of the global model. To mitigate stragglers, which wastes the computing resources of other fast devices, authors in [70] proposed a clustering mechanism named FedHiSyn. All devices are allowed to work at any given point of time which reduces the resource wastage which occurs while waiting for stragglers. FedHiSyn is a tiered federated learning framework which combines both centralized and decentralized federated learning frameworks and clusters devices based on their computing capabilities. After the devices are clustered, the local models trained are shared with the next devices in a ring topology before the final model can be uploaded to the server for global model formation. Therefore, allowing local models to be trained more between the devices increases the accuracy of the local model and reduces the communication rounds with the server. The tiered approach proposed in the study is similar to [13] where devices with different computing resources are tiered and model training is performed at each layer.

The authors of [91], consider limited computation power, limited channel bandwidth as the factors that causes the emergence of stragglers. Similar to [99] where client scheduling is performed and a timer is fixed for the server to wait for straggling clients before proceeding with the global model aggregation, the threshold bases device selection is chosen to limit the impact of stragglers on training performance. The authors consider two basic aspects that can be utilized to deal with stragglers. They are 1) pruning ratio and 2) wireless channel conditions. The pruning ratio helps in evaluating the computing power of a local participating device, whereas the channel conditions help in finding stragglers in the system. It is concluded that if a device has larger pruning ratio values, it is more susceptible to becoming a straggler. However, once obtaining the pruning ratio for all the participating devices, devices with larger pruning ratio values can be removed to reduce the model aggregation error. Therefore, despite achieving a global optimal model due to lower computational overhead, the pruning ratio, allocation of resources to the end devices, and device selection is not effective nor does it help to achieve a globally optimal model. Further, to minimize the device associated cost, the authors in [211] proposes a reinforcement learning based algorithm for training optimization. Similar to the above defined algorithms, the proposed algorithm also focuses on client scheduling. Hence, scheduling is now the most prevalent approach for optimizing and making federated learning efficient in the presence of stragglers. In [176], both system and statistical heterogeneity are counted as the source of impacting the model efficiency. The main role in this study is that of a scheduler which determines the distribution type across participating clients. Scheduling decisions are made based on the initial distribution information provided by participating devices at the beginning of the model training. The steps are as follows: 1) the participating device connects and shares a brief summary of the distribution information to the server; 2) after receiving the information from the devices, the server compares all summaries to identify devices with similar distribution and clusters them as a group. Within these clusters, the fastest devices are scheduled to perform the training rounds; 3) upon selection, the fastest devices train the local model and share updates with the server for global model aggregation. Hence, the client scheduling strategy has been the most commonly used in federated learning to improve training efficiency. However, scheduling can be just a part of the list of approaches involved for improved training efficiency. Recently, coded computing has been proposed to deal with the delay associated with stragglers. The proposed algorithm is termed the straggler-resilient FL scheme, [180]. The main idea is to replicate some amount of data across all participating clients while maintaining the same privacy level compared to the traditional federated learning. The proposed scheme comprises of three phases: They are as follows: 1) encryption; 2) data Sharing; and 3) computation for federated learning. In the encryption phase, the code MDPC is used and row-wise encryption is performed. For the data sharing strategy, once the edge device completes local training and returns its results after a threshold timing, the server alerts the participating device with an error and allows it to share encrypted data to the faster worker. In the data sharing phase, the server monitors the performance of the participating edge devices for all the training epochs. If at any given point of time, a device responds after a certain threshold, an error exception is sent to that device asking it to encrypt the data and send it to the faster device which can resume the process and can remove the hurdle of node failures and its effect on the training performance. Hence, data is shared with the fast devices in the encrypted format and the performance is improved by not halting the progress of a task in between the training process. During the computation phase, the server decodes the values from the edge devices and aggregates it into the final model. Additionally, when the information is decoded if there are any error locations, then erasures are applied and model aggregation is performed afterwards. Apart from stragglers originating from computation overhead, stragglers from network overhead are considered in [46] by proposing the iSample algorithm. Therefore, the client selection parameters are limited as compared to network parameters. In iSample, there are two aspects of data flowing - a model which consists of millions of parameters and selection parameters which are fewer compared to model parameters. Therefore, machine learning model transfer depends on the network throughput whereas the selection parameters depends on network latency. iSample is designed to handle devices with low performance to provide high training efficiency by selecting the most efficient participating devices. Therefore, the clients considering both computation heterogeneity as well as the network overhead are evaluated. The main difference with iSample and benchmarks is, the latter uses the client selection step before training and the client selection is done beforehand. However, iSample makes sure that for subsequent training rounds, the aggregated model is sent only to those clients who are identified or qualified to be the better one. iSample achieved an improved training performance by 27% to 39% compared to the benchmark algorithm, federated averaging. This study is similar to [102] where FedCS is used for efficient client selection to improve training performance and mitigate stragglers.

The authors of [213] adopt theformation of two different groups for federated training. The first group is where the inexact ADMM is used to update the model parameters and the second group is where the server can put stragglers so such that they avoid the training process which will reduce its impact from training and improve efficiency. This method is a good choice to improve training efficiency since stragglers are removed and kept it in a separate group. However, the training accuracy can be affected since any valuable information from the stragglers is avoided completely. This approach is similar to the benchmark algorithm, federated averaging, which avoids the incorporation of stragglers from the training process. Adaptive optimization is focused more in [50] since it has improved the performance of training in the literature and reduced the training associated with communication and computation costs. They proposed the theoretical principle and designed adaptive optimization techniques for federated settings. The authors of [38] proposed coded offloading techniques to improve training efficiency. Offloading techniques improve the overhead associated with computation.

In [103], ClusterFL is proposed which gives high system performance with improved accuracy and reduced communication latency associated with stragglers. Further, guidelines on choosing the hyper-parameters for training are provided. In addition to this, ClusterFL has two mechanisms: 1) cluster based straggler removal; and 2) worker node selection based on the correlation. That is, stragglers are dropped from within a cluster once they are identified by the server. The results show that the method of identifying stragglers among all the working nodes is less effective than the one where stragglers are identified within a cluster and then removed from training. Similarly, correlation-based worker node selection, selects the participating nodes only if the correlation is high within a cluster. Hence, this will increase collaborative learning. If for instance, a worker node in a cluster 1 and another worker node in cluster 2 have weaker correlations with other nodes within the cluster, then removing those worker nodes during the intermediate phase of training can cause accuracy drop only within the same cluster. Therefore, there will only be minor degradation within the cluster with respect to cluster accuracy.

To accelerate federated learning in edge computing, the authors of [172] proposed FedCH for heterogeneous settings. The main idea behind FedCH is that it determines the optimal number of clusters with resource constraint participating edge nodes and forms cluster topology for hierarchical aggregation. Further, to extend FedCH to more dynamic setting, the authors proposed two approaches as follows: 1) fixed re-cluster formation; and 2) an adaptive re-cluster formation. In fixed re-cluster formation, all participating nodes are clustered after fixed number of epochs. However, this approach has to wait for an epoch threshold before clustering can proceed. Therefore, it cannot adapt itself to changing network conditions nor any sudden changes to network status due to stragglers. Due to this issue, another approach known as adaptive re-cluster formation is proposed which is more flexible compared to fixed re-clustering. It monitors the condition of each cluster and identifies if any cluster has stragglers based on its performance. If stragglers are identified, then the time for aggregation between two consecutive global aggregation rounds is increased such that sufficient time is given to stragglers for their contribution in global model aggregation. Hence, the latest algorithm is known as Dyn- FedCH. Hence, the real time delay associated with computing as well as communication delay together with corresponding information for cluster formation is introduced. The system challenges of participating edge devices in federated learning is investigated in [107]. The system challenges include fault-tolerance or straggling-effect due to failed worker nodes in the system. To deal with stragglers in heterogeneous settings, a synchronous scheme for model aggregation is adopted.

The authors of [58] introduces the heuristics for stragglers in federated edge networks. The main idea is that, in the aggregation phase, some of the information only from the fastest worker nodes are utilized in aggregation. Further, this can be performed for local as well as network wide global aggregations. Semi-synchronous scheme for federated learning is proposed in [137]. The main idea is to make convergence faster with reduction in communication as well as energy associated costs thereby mitigating straggling effect, particularly in heterogeneous settings with computational overhead. The proposed scheme is known as SemiSync, defining a synchronization time as to when all the participating workers can share their local models and obtain the global community model. We can achieve faster convergence by fixing a synchronization point such that the data processed by each participating worker does not become contradictory. Further, compared to the synchronous synchronization schemes for federated learning, participating workers do not remain idle or waste their computation resources thereby improving the efficiency degraded by stragglers. That is, the synchronization point is independent of epoch completion time. Since, the approach utilizes batch-processing, it allows more flexibility in obtaining fine-grained control over what is a contribution to the global community model. Hence, SemiSync seeks to balance the communication costs and better resource utilization which deteriorates in the presence of stragglers. In the study in [178], the focus is on node selection for federated training. The authors proposed a probabilistic node selection framework known as FedPNS to change the probability of each node selection dynamically. This change in probability depends on the output of optimal aggregation where a subset of participating devices is obtained reflecting the optimal subset of local updates of the already participating local devices. Hence, these nodes which previously helped in faster convergence have a greater probability of being selected in the subsequent training round's node selection. The main idea in SemiSync is that a node selection policy is identified with respect to the data heterogeneity of participating workers to identify nodes which can improve model convergence.

SemiSync calculates the product of local and global updates which eventually reflects the difference in distribution between the nodes as well as the overall distribution. The nodes whose distribution can affect the local updates and convergence are identified and excluded from being picked up for the training rounds. That is, participating nodes with a higher possibility of making local updates worse and degrading convergence get lower-to-no probability of being selected for federated training. Therefore, only the nodes with a high possibility of decreasing the global loss have a high probability in its selection. Hence, this can speed up the convergence compared to the benchmark algorithm, federated averaging.

Chapter 4

Proposed Research

Overview

Edge federated learning faces many challenges, such as massive client distribution, system and statistical heterogeneity. Of all the challenges, system heterogeneity causes differences in the computational timings of participating edge devices while statistical heterogeneity also has a negative impact on training accuracy and efficiency. These heterogeneities cause a straggling effect where the edge devices slow down the training efficiency due to their computational and statistical variances. Our aim is to focus on system and statistical heterogeneities arising from the limited computational power of edge devices and the large number of parameters from high dimensional data, since federated training involves thousands of devices, all of which have some form of heterogeneity. Hence, training with the current benchmark algorithm, FedAvg [81] with the aforementioned challenges of stragglers can reduce training efficiency. In FedAvg and most of the algorithms for federated learning, the server waits for all the de- vices for model aggregation. Therefore, the delay increases and efficiency decreases. In this chapter, we obtain an optimal solution by introducing a balance between system and statistical heterogeneity where information from the stragglers is incorporated in the training without ignoring them, as in FedAvg. Furthermore, we improve the training efficiency by reducing the number of parameters involved in each round of the training process. We introduce the elastic optimized distributed algorithm for stragglers in edge federated learning. We discuss the problem of stragglers in edge federated learning and their mitigation, aiming to improve training performance in terms of accuracy over the communication rounds. We set the percentage of stragglers as 10% and 50% of the edge devices during federated training, as a realistic scenario. Since we consider stragglers as 10% or 50% during federated training, we mean the this means that 10%/50% of edge devices do not perform all training rounds but submit their partial results at the end of the training process, thereby performing a random number of iterations in each iteration. Further, the reduction in the number of parameters speeds up the training process.

Heterogeneity and convergence: As previously mentioned, we consider system and statistical heterogeneity in our work. The incorporation of system heterogeneity implies that devices which cannot complete their required number of local iterations in training can perform any random number of iterations and contribute to model aggregation. High dimensional data with a larger number of parameters is a long-standing statistical problem in the literature which contributes to statistical heterogeneity causing the straggling effect. The divergence of edge devices from the local objective is due to statistical heterogeneity which worsens when the number of model parameters is very large and when computationally-inefficient straggling devices contribute their incomplete results in model aggregation. In the literature, different notations are used to specify the intensity or degree of heterogeneity as per the variances in the local objective assumptions. For instance, in [81],], the bounded dissimilar gradient assumption is analyzed, whereas the work in [189] first used the term gradient diversity which measures the dissimilarity degree of individual gradients of the loss functions. Therefore, the gradient is larger when the products between the gradients are small. Its ratio is defined as follows

$$\nabla(\omega) = \frac{\sum_{i=1}^{N} ||\nabla f_i(\omega)||_2^2}{||\sum_{i=1}^{N} \nabla f_i(\omega)||_2^2} \tag{4.1}$$

4.1 Methodology:

In this section, we describe the model developed in our research and present the mathematical details of the regularized stochastic gradient descent.

4.1.1 Logistic regression:

Logistic regression is the most common technique used in statistics to explain the relationship between dependent and explanatory variables. Similar to other regression analysis techniques, logistic regression is used for predictive analysis. It helps in describing the data and its relationship between one independent variable and one or more existing nominal independent variables. It can describe the connection between the output predictor, y and the input feature, x, such that the best fitting solution for the model parameters is obtained by minimizing the equation as follows:

$$f(x) = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{4.2}$$

where y_i is the i-th output predictor and x_i is the i-th input feature. The best fit line for linear regression can be formulated as follows:

$$y = \gamma_0 + \gamma_1 x \tag{4.3}$$

We consider N edge devices represented by n = 1,2,...,N in edge federated learning. We consider logistic regression for image classification. The logistic regression probability has the form of

$$p = \gamma_0 + \gamma_1 x \tag{4.4}$$

Therefore, the model is as follows:

$$log \frac{p}{1-p} = \gamma_0 + x\gamma_1 \tag{4.5}$$

where γ_1 and γ_0 are coefficients. X is the vector of parameters.

Let us apply an exponent on both sides of eq (4.4):

$$exp[log\left(\frac{p}{1-p})\right] = exp(\gamma_0 + \gamma_1 x) \tag{4.6}$$

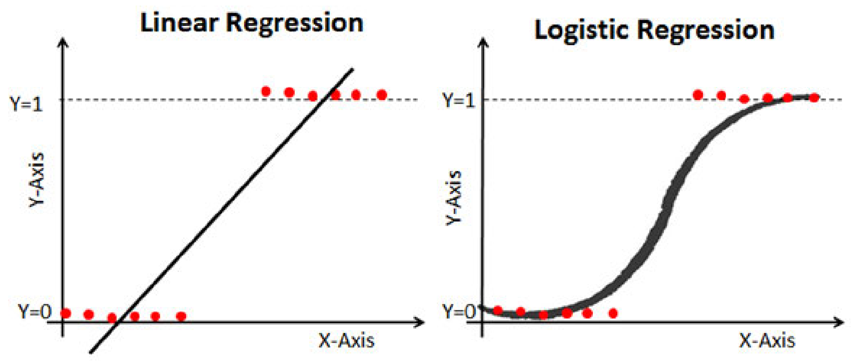


Figure 4.1: Linear and Logistic regression curves
source: www.datacamp.com

But, to compare of the label probabilities, the above function is not suitable. Therefore, to output the values in the range of [0,1] for the classification probabilities identified, the sigmoid function can be used. The sigmoid function g for a function $f(x)$ can be written as follows:

$$g(f(x)) = \frac{1}{1+e^{-f(x)}} \tag{4.7}$$

The following hold for the sigmoid function:

$$Prob[Y_i = positive] = \frac{1}{1+e^{-f(x)}} \tag{4.8}$$

$$Prob[Y_i = negative] = 1 - \frac{1}{1+e^{-f(x)}} \tag{4.9}$$

where the probability is either +1 or −1.

The coefficients are obtained by maximizing the log likelihood with an elastic penalty imposed on the L1 and L2 norm of the coefficients for logistic regression. Therefore, we can write the penalized logistic regression loss function as follows:

$$l(\gamma_0, \gamma, \lambda_1, \lambda_2) = -l(\gamma_0, \gamma) + \lambda_1|\gamma|_1 + \lambda_2||\gamma^T||^2 \tag{4.10}$$

where l indicates log likelihood, and λ_1 and λ_2 are the tuning parameters controlling shrinkage intensity.

4.1.2 Optimization

The optimization has been as been an interest in solving machine learning problems in literature.

As described in earlier chapters, stragglers emerge for various reasons such as statistical heterogeneity and system heterogeneity. We incorporate system heterogeneity in our framework to exploit the benefits of data from straggling devices with varied computational capabilities. However, since these heterogeneous devices cannot complete all training rounds, they are allowed to submit the results from a random number of rounds as per their capabilities. Hence, we call this heterogeneous partial results. Further, the heterogeneous partial results submitted causes gradient divergence as the local minima is difficult to find because of the full and partial results submission.

Stochastic Gradient Descent:

An optimization problem generally involves the minimization of a mathematical function. For instance, for a given function $f(x) = x^2$, an optimization problem involves finding the value of x which can minimize the function. To learn optimal model parameter values ω, gradient descent is a prevalent strategy adopted in the literature. The objective function can be minimized by using a number of steps in the negative gradient direction

$$\omega^{t+1} \leftarrow \omega^t - \eta\phi_i(\omega) \tag{4.11}$$

where ϕ is the learning which can be a function of time. When ϕ decreases, it ensures that the model will converge within a distance of ϵ.

Local SGD:

A very popular methods for solving distributed optimization problem of type 4.12 is known as local stochastic gradient descent (Local SGD). Each participating device performs gradient steps locally and returns the local updates to the node performing final aggregation which could be done by a central server or a device within the collaboration. Generally, when the number of local steps are fixed with local SGD, training performance improves. Therefore, a fixed number of local steps can improve communication efficiency. Further, if workers do not communicate in the local SGD then Local SGD performs even more better as communication is reduced. According to [65], local SGD is a critical component of federated learning. Further, the addition of a regularization term in local SGD can improve generalization [204]. Local SGD applies stochastic gradient descent as a local optimizer. However, there is no guarantee that using local SGD can guarantee convergence without any assumptions. Therefore, the assumption of the gradient bounds and dissimilarity is made in most of the works in the recent literature. For instance, in [80], gradient dissimilarity is assumed and the bound is made on the gradients. Generally, the common assumptions of Lipschitzness, $\|\nabla F_j(\omega) - \nabla F_j(\omega^{|})\| \leq L\|\omega - \omega^{|}\|$, is assumed for any of the theoretical guarantees followed by bounded and dissimilar gradients.

Federated Optimization:

Generally, for any ML algorithm, the critical aspect of optimization is to improve accuracy and reduce the number of rounds required to reach target accuracy [24, 27, 54]. The number of communication rounds in federated learning plays a crucial role as the rounds are usually higher in number in federated learning. Therefore, the larger the number of communication rounds, the longer the server would be idle, waiting for the responses from the edge devices. Optimization in federated learning has many issues associated which is unlikely to occur in the centralized machine learning. Some of the common characteristics are as follows:

- Data heterogeneity: Data stored across the clients is never directly communicated with the server or with other clients as data privacy is one of the basic aspects of federated learning. Clients have different quantities of data that they hold. In traditional distributed settings, data is distributed among the participating clients and various techniques are used to make sure that the data is balanced among all the clients to achieve training efficiency. Data heterogeneity is a critical aspect in federated optimization as it accounts for training efficiency as well as accuracy of the global model. Therefore, dealing with data heterogeneity in federated learning training is of paramount importance.

- Communication inefficiency: One of the critical issues is communication inefficiency when the clients cannot communicate with the server due to network bandwidth or associated heterogeneities. This has resulted in the requirement of communication efficient optimization algorithms for federated learning. Standard federated learning averages over many local model updates and it still achieves efficiency with respect to communication between the participating clients and server as well as obtaining global collaborative model similar to the model obtained in traditional centralized training. However, the standard algorithm does not consider stragglers in the training process. It simply ignores stragglers and allow non-stragglers to be involved in federated training. The main idea is to identify how to reduce the number of communication rounds that are required for a model to converge. One way is to allow participating clients to allow more local iterations or training rounds so that the local updated model can then communicated with the server. Therefore, the number of communication rounds can be reduces and training can be made efficient. However, allowing straggling devices to perform more local iterations can cause gradient divergence thereby causing accuracy degradation.

- System complexity and associated heterogeneity: Usually system heterogeneity refers to the compute and bandwidth resources differences among the participating clients in federated learning. System as well as data heterogeneity can heavily impact on training time. We can associate data heterogeneity with the following: 1)distribution of training samples across each participating client; 2) feature distribution among the clients. For federated learning, on-device hardware capabilities alone can play a crucial role as the training speeds depends on the system capabilities of the devices. Further, the participating client should be able to handle large number of parameters associated with model training. If the memory of the client is not sufficient to handle a large number of model parameters during training, it can further degrade system efficiency causing a deadlock. One solution can be the use of transfer learning which allows the use of an already trained model which does not require the model to be trained from scratch. Hence, a generic model can be further trained to become more specific with less training thereby improving system efficiency. In addition to this, client-sampling can also play an important role as better client selection can avoid training inefficiency. Further, resource allocation techniques can allow resource constrained devices to perform training efficiently.

4.1.3 System Modelling and Analysis

In this section, we present our proposed model FedEN (Elastic Optimized Federated Learning) which is a customized version of FedAvg and FedProx. Secondly, we incorporate the contribution from stragglers. We set the percentage of stragglers as 10% and 50% of the edge devices during the training process. We describe the general edge federated learning framework, the stragglers in the proposed framework and introduce the straggler-resistant optimization algorithm. We first describe the learning based on edge computing and federated learning to propose edge federated learning in the subsequent section. In edge computing [133],to train an ML model [170], data is offloaded at the edge server. The edge server then trains the offloaded data from the edge devices. The benefit of this type of data offloading is that the data is easily accessible rather than downloading data from the centralized server. Further, the communication time with the centralized server is greatly reduced with the introduction of a server at the edge level. Bandwidth is improved and latency is decreased. In federated learning [96], the clients utilize their local datasets and perform local training instead of transferring the data to the server for training. The main advantage is ensuring the privacy of the local data as it is not transmitted elsewhere for training.

4.1.4 Edge Federated Learning

Edge federated learning is a term used for federated learning utilizing edge computing. That is, federated learning is implemented at the edge of the network thereby reducing data migration to a centralized location preserving privacy. Heterogeneity will remain a challenge again in edge federated learning as most of the edge devices are resource constraint with different computational capabilities and differences in the distribution of data they hold as compared to the overall population. In addition to this, the data edge devices hold depends on their environment. The system heterogeneity can still be identified based on the training performance. However, the data or statistical heterogeneity is difficult to identify. Consequently, designing efficient federated learning algorithms needs the consideration of resource constraints as well as the heterogeneities associated with system and client's data. There are various aspects which needs attention in edge federated learning which are defined as per [149]:

- Proper Device Selection: This refers to the selection of devices that will participate in model training in edge federated learning. In the benchmark algorithm, random clients are selected for federated training. However, since stragglers are avoided in benchmark algorithm, the selection of appropriate edge devices can reduce the overhead of training latency in the context of edge federated learning.

- Appropriate Resource Allocations: For optimal training efficiency, appropriate resource allocations can be the solution. Meta heuristics can be considered to help with computational complexity associated with changing edge federated learning environment.

- Aggregation process: The communication and aggregation frequency are important as it can reduce the number of local computations required. The benchmark algorithm, federated averaging, uses weighted average of all the local updates received and follows a synchronous approach for synchronization. However, the synchronous approach is not beneficial when considering stragglers as it delays the aggregation process. Different to the standard synchronization method used, asynchronous methods have been adopted to incorporate delay associated with stragglers and allow faster aggregation. However, asynchronous methods can include stale gradients which deteriorates accuracy to a great extent. Hence, appropriate combination of synchronous and asynchronous methods can be adopted for faster aggregation with improved accuracy.

- Local learning optimization: It is important that all the participating devices in federated training have identical objectives to achieve a better collaborative model.

- Optimization of time and energy: Time required for obtaining a global collaborative model depends on the devices participating in the training rounds as to how better clients are in terms of their system computational power as well as their data distribution. Scheduling of clients plays an especially significant role in optimizing the time and energy consumption for federated training.

From the perspective of performance, the aforementioned two schemes are limited by either a computation or communication barrier. In edge computing learning, offloading data to the edge server can cause privacy issues. Further, the data uploading time is relatively high if the datasets are large. On the other hand, the parameter server architecture of FL is rigid and creates a bottleneck when communicating the training results and when downloading the initial model parameters.

We investigate an edge federated learning context which consists of an edge server and N edge devices available for federated learning (FL). These N edge devices can be represented as $N = \{1,2,...,N\}$. Only $S \subset N$ are randomly selected for the training process. Let T be the communication rounds required for model aggregation and the index for representation be t. E is the total number of updates performed by the edge devices for local training with the index e. The model parameter is denoted as ω. For a edge device i in round t with local update step e, the model parameter is represented as $\omega_{i,e}^{t}$.

The main objective for efficient EFL is to minimize the loss which can be formulated as follows :

$$min \frac{1}{N}\sum_{i=1}^{N} f_i(\omega) \tag{4.12}$$

where $f_i(\omega)$ is the loss function for device i for the sample (x_i, y_i).

For some j edge devices, the loss function is:

$$F_j(\omega) = \frac{1}{N_j}\sum_{i\in\beta_j} f_i(\omega) \tag{4.13}$$

The global loss is defined as the summation over all the local loss functions :

$$f(\omega) = \sum_{j=1}^{J}\frac{n_j}{n}F_j(\omega) \tag{4.14}$$

The optimization problem in 4.12 can cover linear regression or logistic regression and also complicated models such as neural networks. For any given pair of input (x, y) and with a loss function, f, some of the examples are as below:

- Linear regression

$$f(\omega) = 1/2(x^T\omega - y)^2, \forall i \in N \tag{4.15}$$

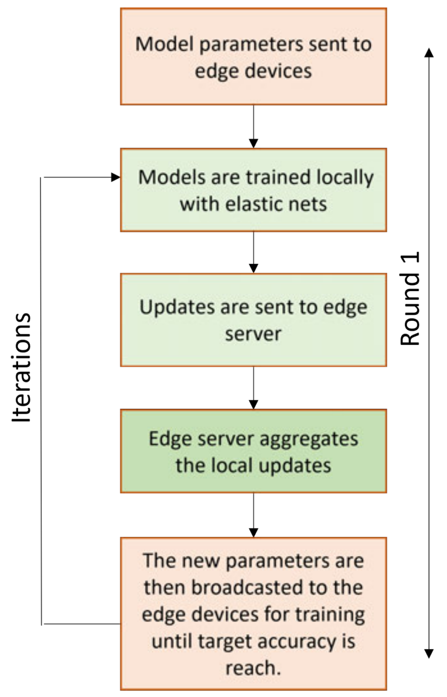

Model parameters sent to edge devices
Models are trained locally with elastic nets
Updates are sent to edge server
Edge server aggregates the local updates
The new parameters are then broadcasted to the edge devices for training until target accuracy is reach.
Iterations
Round 1

Figure 4.2: Edge federated Learning workflow

- Logistic regression

$$f(\omega) = -log(1 + exp(-yx^T\omega)), \forall i \in N \tag{4.16}$$

Generally, more complicated models for convex and non-convex problems in the context of neural networks where the predictions are made through non-convex function with gradients being computed using backpropagation methodology. For federated optimization, it must try to handle the data for training with the characteristics listed below:

- Massive Distribution of Clients: The amount of devices which could participate in federated training are massive.
- Unbalanced data: Different participating devices in federated training might have unbalances training samples. That is, the amount of training data differs among the devices.

Generally, there are three steps in edge federated learning training as follows: a) the edge server broadcast the model parameters server to the clients; b) clients train and update the local model and the model is sent to the edge server; c) the edge server aggregated the model and steps (a) to (c) are repeated until target accuracy is reached. The local update performed by the clients can be modelled as follows:

$$\omega_i^{t+1} \leftarrow \omega_i^t - \eta\phi_i(\omega_i^t) \tag{4.17}$$

where ω_i^{t+1} is the model parameter for edge device i for rounds $t+1$. After E local model updates as in eq(2), the updated parameters are then aggregated by the edge server as follows:

$$\omega^{t+1} \leftarrow \omega^t + \frac{\eta_\phi}{|S|}\sum_{i\in S}(\omega_i^t - \omega^t) \tag{4.18}$$

The aggregated model parameters then form a global model whose model parameters are again broadcasted to the edge devices in an iterative manner.

Algorithm 2: Federated Averaging

```
Input: B, N, E, $\eta$
Output: federated trained global model
while target accuracy not achieved do
    Global model parameters broadcast to edge devices;
    Local model training at each client;
    if complete local training after E updates then
        send the updated parameters to the edge server;
    end
    if local updates received by the edge server then
        perform aggregation using equation(3);
    end
end
```

4.1.5 Elastic optimization

Elastic Net, introduced by [225], is a form of regularization which combines both lasso and ridge regularization linearly. We propose an elastic net in the optimization of federated local objective functions to produce sparser and better generalized models which can deal with the emergence of stragglers due to large number of parameters of high dimensional edge data and diverged parameters.

Let ω be the model parameter, x be the input feature, and y be the predictor for the input feature. Taking expectation of the loss function, the new loss is represented as follows

$$E_j[l(\omega, x, y)] = l(\omega, x, y) + \pi(\omega) \tag{4.19}$$

And

$$\pi(\omega) = \lambda\left[(1-\alpha)/2||\omega||^2 + \alpha||\omega||\right] \tag{4.20}$$

Elastic net is particularly useful when the data set is high dimensional with predictors(p) being larger than the observations(n). Elastic net has a tuning parameter, λ and a mixing parameter, α. The mixing parameter, α, regulate the amount of weight given to lasso and ridge. When $\alpha = 0$, elastic net is equivalent to ridge regression. Similarly, when $\alpha = 1$, elastic net penalization becomes special case of lasso thereby imposing only $l1$ penalty on the model parameters while training. $l1$ penalty helps in better feature selection thereby giving sparse results. However, the $l2$ penalty adds smoothness to the objective function by shrinking the model parameters. Therefore, ridge penalty shrinks the model parameters so that the gradient divergence is reduced and local optima is achievable. Despite being similar to the benchmark algorithms, FedAvg and FedProx, our proposed algorithm, as shown in Algorithm 3, FedEN, has the following difference:

- A fraction, C, of edge devices are selected for the training process
- Each edge device downloads a shared model, and uses their own dataset to contribute in the collaborative model
- Each participating edge device have a different number of local epochs depending on the percentage of stragglers in the training process.
- All participating edge devices after local training return results to the server to aggregate the collaborative model

Algorithm 3: FedEN:Elastic Optimized Federated Learning

```
Input: B, N, E, $\eta$
Output: Trained global model
Edge Device: i = 1,2,...,N:
while target accuracy not achieved do
    Global model parameters downloaded to edge devices;
    Local model initialization and training using Eq4.19 ;
    $y_i \leftarrow \omega$
    $\omega_i^{t+1} \leftarrow \omega_i^t - \eta_i \phi(\omega_i^t) + \pi(\omega_i^t)$
    if local training complete after E steps then
        send the updated model parameters from model $y_i$ to the edge
          server;
    end
    Edge Server:
    if local updates received by the edge server then
        perform aggregation using Eq4.18 ;
    end
end
```

Selection of α: This experiment has the effect of mixing parameter α on the performance of the proposed algorithm FedEN.

The effect of mixing parameter on the performance of FedEN: When the mixing parameter is between 0 and 1 ($0 \leq \alpha \leq 1$), the test accuracy is affected and fluctuates. Training efficiency increases with larger values of the mixing parameter while smaller values are suitable for achieving optimal convergence.

4.2 Theoretical Assumptions and Convergence Analysis:

We investigate the objective function, the straggler optimization problem, as previously discussed in our convergence guarantees. We incorporate typical assumptions on ML models for theoretical analysis. The following are some of the examples of assumptions:

we employ the following standard assumptions ([81], [96]) for edge federated learning algorithm:

Assumption 1 (*Lipschitz gradient*) : The Function f is L-smooth if its gradient is L-Lipschitz continuous, i.e.,for any two model parameters, ω, $\omega^{|}$, $F_j(\omega)$ is the Lipshitz gradient for each edge device j such that $j \in 1, 2, ..., N$. That is, $\|\nabla F_j(\omega) - \nabla F_j(\omega^{|})\| \leq L\|\omega - \omega^{|}\|$.

Assumption 2 ($B - Gradient\ Dissimilarity$: The local gradients are B-dissimilar from each other. That is, $\|\nabla F_j(\omega)\| \leq B\|\nabla f(\omega)\|$

Assumption 3 ($\beta-$*Inexact Local Solutions* For a each device j such that $j \in \{1, 2, ..., N\}$ and t communication rounds, local training results in β dissimilar solutions, i.e., $|\nabla f_j(\omega_j^{t+1}, \omega^t)| \leq \beta|\nabla f_j(\omega, \omega^t)|$.

Assumption 4 $BoundedGradients$: For all model parameters, $\omega \in W$ and the regularizing parameters in $\pi(\omega)$, we have bounded gradients assumed to be $|\nabla F(\omega, x)|^2 \leq B_\omega$.

Assumption 5 ($BoundedVariance$) Let $\nabla F(\omega, X)$ be the stochastic gradient for X, and a function $f(\omega, x, y)$

Assumption 6 ($BoundedDomain$) : Assume the model parameters are from devices from the same domain thereby having most of the data coming from same distribution.

General convex loss function The loss function in 4.12 is strictly convex and differentiable. For any prediction matrix, A, sub-gradient, Γ, lasso solution, $\beta^{`}$, the KKT conditions for 4.12 can be presented as follows:

$$A^T(-\nabla f)(A\beta^{`}) = \lambda\Gamma, \Gamma_i \in \begin{cases} sign(\beta^{`}), & \text{if } \beta^{`} \neq 0 \\ [-1, 1], & \text{if} \beta^{`} = 0 \end{cases} \tag{4.21}$$

Where ∇f is the function of R^n. We can define the equicorrelation set and signs as follows:

$$\begin{aligned} \xi = \{i \in \{1, 2, ..., p\} : |A_i^T(-\nabla f)(A\beta^{`})| = \lambda\}, \\ s = sign(A_\xi^T(-\nabla f)(A\beta^{`})) \end{aligned} \tag{4.22}$$

If $A \in R^{nxp}$ is a prediction matrix with entries drawn from the continuous probability distribution on R^{np}, then for a strictly convex function, f, which is differentiable and $\lambda > 0$, the objective function in 4.12 has a unique solution with probability as 1. More generally, the results can be applied to any convex, differentiable loss function, f, as in the following logistic regression loss as follows:

$$f(\omega) = \sum i = 1n\{-y_i x_i + lof(1 + exp(x_i))\} \tag{4.23}$$

A local minima of a convex objective function is also a global minimum. Let ω^* be the $\epsilon - approximation$ optimum when the following holds:

$$\forall_{\omega \in W}, \exists\ f(\omega^*) \leq f(\omega) + \epsilon. \tag{4.24}$$

The minimum of a function which is differentiable at a point is where the derivative is zero as in the following:

$$\nabla f(\omega) = 0 \tag{4.25}$$

For a constrained optimization problem, the point where the model parameter has minimum value is where the inner product between : negative gradient and the direction towards the W is not positive. This is supported by the KKT optimality which considers more than one dimension of a function obtaining the minimum such that the negative gradient points outwards.

Theorem 1 (Karush-Kuhn-Tucker): For any $\omega \subseteq R^d$ and $\omega^* \in min_{\omega \in W} f(\omega)$. Then for any $y \in W$, it is said that

$$\nabla f(\omega)^T (y - \omega^*) \geq 0. \tag{4.26}$$

Lemma 1: Local SGD can be defined as follows:

$$\omega_{t+1} = \frac{1}{N} \sum_{i=1}^{N} \omega_t - \eta_t \nabla f(\omega_t) \tag{4.27}$$

Theorem 2: Let assumptions 1, 4, 5 hold. Then each round of convergence of the optimized SGD with step size $\eta_t \leq 1/2l$ has the following properties:

$$E[f(\omega_t) - f(\omega_{t+1})] \geq E[\frac{\eta_t}{2} |\nabla f(\omega_t)^2|] - \eta_t \sigma^2 \tag{4.28}$$

The convergence results with a variance zero, $\sigma = 0$, step size, $\eta_t = 1/2l$ is given by [119] as $\mathcal{O}(lB/T)$. The amount of variance affects the convergence time. The larger the variance, the longer it takes to reach convergence time. Therefore, the use of an elastic term remediates this effect by utilizing the elastic optimization approach which improves prediction accuracy and training efficiency by reaching target accuracy in fewer communication rounds.

Theorem 3 (ϵ- optimal stability): An algorithm X(.) is ϵ optimal stable if the datasets $D_1 = \{d_1, d_2, ..., d_n\}$ and it twin $D_2 = \{d_1^{'}, d_2^{'}, ..., d_n^{'}\}$ with d being the data test sample have the following:

$$E_X[|f(X(D_1); d) - f(X(D_2; d))|] \leq \epsilon \tag{4.29}$$

Therefore, $\epsilon-$ stability indicates that the generalization is bounded by ϵ.

Theorem 4 (Smooth Functions): Let $F(\omega)$ be a function which is $\gamma-smooth$ with step size, $\eta = \frac{\epsilon}{\gamma N^2}$ and T be the number of communication rounds. Then, after $\frac{2}{\epsilon^2}N^2\gamma(F(\omega)-F*) \leq T$ updates with $(1+\delta/2)\epsilon \geq E[||\nabla F(w_i)||_2^2]$ where $N^2 = 1/n\sum i = 1n||\nabla f_i(\omega)||_2^2$, $\epsilon-$ *optimality condition*, δ *is a constant.*

Theorem 3 and 4 have also discussed in [189] where the authors discuss the informal convergence guarantees for gradient diversity to achieve an $\epsilon-$ optimal solution by bounding the distance to the optimal by B times.

Theorem 5: Let Assumption 2 hold. That is, $E\|\nabla F_j(\omega) - \nabla f(\omega)\|^2 \leq \sigma^2$. For any ϵ, we have $\sqrt{1+\frac{\sigma^2}{\epsilon}} \geq D_\epsilon$. We follow the assumption of dissimilarity as stated in [81]. We derive the relationship between gradient dissimilarity for the local objectives and bounded variance for the gradients as follows:

$$E_j\|\nabla F_j(\omega) - \nabla f(\omega)\|^2 \leq \sigma^2 \tag{4.30}$$

We further expand the equation in (4.30) as follows:

$$E_j\|\nabla F_j(\omega) - \nabla f(\omega)\|^2 \tag{4.31}$$

$$E_j\|\nabla F_j(\omega_{t+1}) - \nabla f(\omega_{t+1})\|^2$$

$$E_j\|\nabla F_j(\omega_{t+1}) - \nabla f(\omega_t) + \nabla f(\omega_t) - \nabla f(\omega_{t+1})\|^2$$

$$E_j\|\nabla F_j(\omega_{t+1}) - \nabla f(\omega_t)| - |\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2$$

The combined equation can be written as follows:

$$E_j\|\nabla F_j(\omega) - \nabla f(\omega)\|^2 \leq \sigma^2 + |\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2$$

We define the dissimilarity, B for $|\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2 \neq 0$ as follows:

$$B(\omega_{t+1}) = \sqrt{\frac{E_j[\|\nabla F_j(\omega) - \nabla f(\omega)\|^2]}{|\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2}} \tag{10}$$

$$\begin{aligned} B(\omega_{t+1})^2 &= \frac{E_j\|\nabla F_j(\omega) - \nabla f(\omega)\|^2}{|\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2} \\ &\leq \frac{\sigma^2}{|\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2} + 1 \end{aligned}$$

Theorem 5 and Assumption 2 quantify the dissimilarity between devices in edge federated learning. The dissimilarity in local functions increases when $B(\omega) \geq 0$. So, a larger B means a larger dissimilarity between edge devices. For all ω there exist ϵ such that $B(\omega) \leq B_\epsilon$. We can say that there is a sub-optimal ϵ solution which differs between the edge devices. When the gradients are bounded by some non-negative constants, we have the following implication:

$$\begin{aligned} B(\omega_{t+1}) &= \sqrt{\frac{E_j[\|\nabla F_j(\omega) - \nabla f(\omega)\|^2]}{|\nabla f(\omega_{t+1}) - \nabla f(\omega_t)\|^2}} \\ &\leq B_\epsilon \\ &\leq \sqrt{1 + \frac{\sigma^2}{\epsilon}} \end{aligned} \tag{11}$$

Theorem 6 (Elastic Convergence) : The elastic net utilized two tuning parameter $\lambda_1, \lambda_2 \geq 0$. When $\lambda_2 > 0$, the solution for the elastic net is unique. We consider the fact that the for any fixed $\lambda_1 > 0$, the elastic net converges to the minimum of $l_2 norm$ lasso solution. By fixing any input from X and $\lambda_1 > 0$, for almost every output predictor, y, the elastic net converges to LARS lasso l_1 norm solution as $\lambda_2 \to 0^+$

Proof: By lemma 13 [155], if for any output vector, $y \notin N$, where $N \subseteq R^n$ the LARS lasso satisfies $\beta^{‘LARS}(\lambda_1)_i \neq 0 \; \forall i \in \xi$. To fix $y \notin N$, we rewrite lasso solution as follows:

$$\beta^{‘LARS}_{-\xi}(\lambda_1) = 0 and \beta^{‘LARS}_{\xi}(\lambda_1) = (A_\xi^T A_\xi)^+ (A_\xi^T y - \lambda_1 s) \tag{4.32}$$

We define the function,

$$f(\lambda_2) = (A_\xi^T A_\xi + \lambda_2 I)^{-1} (A_\xi^T y - \lambda_1 s) for \lambda > 0 f(0) = (A_\xi^T A_\xi)^+ (A_\xi^T y - \lambda_1 s). \tag{4.33}$$

with the equicorrelation set, ξ, and fixed sign, s, the function f is continuous on $[0, \infty)$. Therefore, for small $\lambda_2 > 0$, the elastic net solution for both the tuning parameters is given as follows:

$$\beta^{‘Elastic}_{-\xi}(\lambda_1, \lambda_2) = 0 \; and \; \beta^{‘Elastic}_{-\xi}(\lambda_1, \lambda_2) = f(\lambda_2) \tag{4.34}$$

We show that the above solution satisfies KKT optimality conditions for lasso as in [155], for smaller values of λ_2. Therefore, the KKT conditions for the elastic net problem can be presented as follows:

$$A^T(y - A\beta^{‘Elastic}) - \lambda_2 \beta^{‘Elastic} = \lambda_1 \Gamma \tag{4.35}$$

Where Γ is the subgradient from the KKT optimality condition

We have Γ for i-th subgradient defined as follows:

$$\Gamma_i \in \begin{cases} sign(\beta^{‘Elastic}), & \text{if } \beta^{‘Elastic} \neq 0 \\ [-1, 1], & \text{if} \beta^{‘Elastic} = 0 \end{cases} \tag{4.36}$$

Since, $f(0) = \beta^{‘LARS}(\lambda_1)$ (equicorrelation coefficients of LARS lasso) at the tuning parameter λ_1 and $y \notin N$, the continuity of f for small λ_2. Further, we know $||A^T_{-\xi}(y - A_\xi f(0))||_\infty$. Therefore, we get the following which verifies the KKT conditions for small λ_2 with I being the identity matrix which does not have any inverse:

$$\begin{aligned} & A_\xi^T(y - A_\xi f(\lambda_2)) - \lambda_2 f(\lambda_2) = \\ & A_\xi^T y - (A_\xi^T A_\xi + \lambda_2 I)(A_\xi^T A_\xi + \lambda_2 I)^{-1} A_\xi^T y + \\ & A_\xi^T y - (A_\xi^T A_\xi + \lambda_2 I)(A_\xi^T A_\xi + \lambda_2 I)^{-1} \lambda_1 s \\ & = \lambda_1 s. \end{aligned} \tag{4.37}$$

Further, assuming $\omega*$ is an optimal solution for objective 4.12, then the difference between the optimal function values and the expected function values at some point, ω_t after t iterations is given in [181] as follows:

$$Ef(\omega_t) - f(\omega*) \leq \frac{1}{\sqrt{t}}() \tag{4.38}$$

4.3 Summary

We propose algorithm, FedEN, for optimization of stragglers in edge federated learning. FedEN improves training performance by reaching the target accuracy in less communication rounds. Given the factors that affects the federated training efficiency, distributed elastic optimization is adopted to constraint model from diverging and improving training efficiency. By utilizing the mixing parameter α in the local objectives of participating edge devices, FedEN allows the edge device to use local loss function which is smooth due to ridge regression as well the computational load of large number of parameters is reduced with the help of lasso penalization. Results show that the number of communication rounds required to reach target accuracy is reduced as compared to the benchmark algorithms. In this work, we simulate the training at client side as well at server side by writing modules in pytorch. We consider system heterogeneity in terms of percentage of straggling devices present in the system. We allowed stragglers percentage to be 10%, 50% into the simulating environment. However, in real world scenario the percentage can vary can be more heterogeneous in terms of system capabilities as well as wireless connectivity.

Chapter 5

Experiments and Results

Overview

To achieve our aim, we use logistic regression as the loss function as discussed in section 5.1. To optimize the loss function, we perform elastic optimization of the edge device's local loss function using elastic net regularization

5.1 Datasets

This section includes a brief description of the datasets used for the FL training experiments. Further, a brief comparison of the datasets is given in table below.

MNIST: The Mixed National Institute of Standards and Technology (MNIST) data set is the most popular data set for handwritten image classification. It is publicly available and is used as benchmark data set. The data set consists of gray-scale images of 60,000 training samples and 10,000 testing samples of handwritten digits (0 - 9). The training set consists of handwritten digits from 250 people and the test set contains handwritten digits from different people. Each image is 28X28x1 in size. The intensity of the images is in the range 0 (black) to 1 (white).

CIFAR-10: The data set consists of 50,000 training samples and 10,000 testing samples. These are color images sized 32x32. The images are of 10 different classes. CIFAR-10 images have been cropped to 28x28 in size and trained on CNN as per the specification shown in table below:

Dataset	Edge devices	Train samples	Test samples	Train rounds
MNIST	100	60000	10000	200
CIFAR-10	100	50000	10000	200

Table 5.1: Data sets

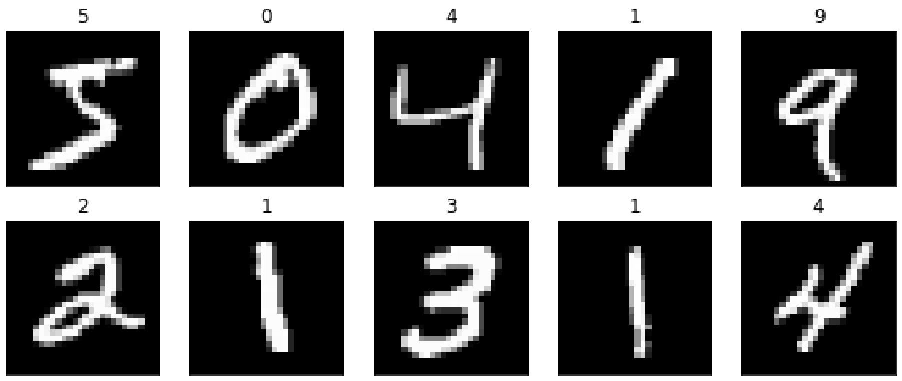


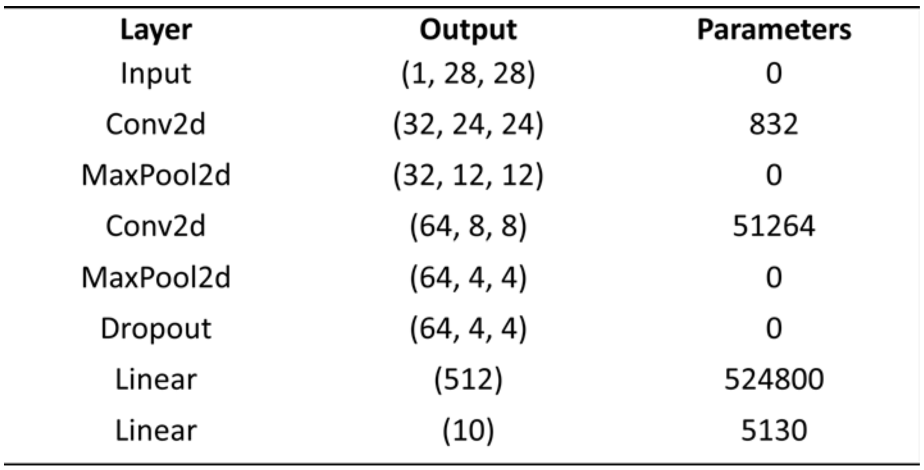
Figure 5.1: Mnist data sample observations

Figure 5.2: CIFAR10 data sample observations

CIFAR-10 is trained on CNN with specifications provided in training FedAvg. The CNN for CIFAR-10 has two convolutional layers with 64 5x5 filters followed by two fully connected layers with 394 and 192 neurons.

The network used is a convolutional neural network with the specifications used in [96]. The full network architecture for MNIST and CIFAR-10 is shown in the Figure 5.3. Each architecture consists of two convolution layers followed by the max pooling layers. ReLu is the activation functions with soft max at the final layer for classification. Further, the output and number of parameters at each layer is also specified.

Data Partitioning: The data are partitioned among the participant edge devices so that each device gets at least two of the same class samples for local training. Modules which performs data partitioning are defined so that the edge device or the edge client receives data from only some subset classes and doesn't



Layer	Output	Parameters
Input	(1, 28, 28)	0
Conv2d	(32, 24, 24)	832
MaxPool2d	(32, 12, 12)	0
Conv2d	(64, 8, 8)	51264
MaxPool2d	(64, 4, 4)	0
Dropout	(64, 4, 4)	0
Linear	(512)	524800
Linear	(10)	5130

Figure 5.3: MNIST CNN

Layer	Output shape	#trainable params
MNIST	100	60000
CIFAR-10	100	50000

Table 5.2: MNIST CNN Network Architecture

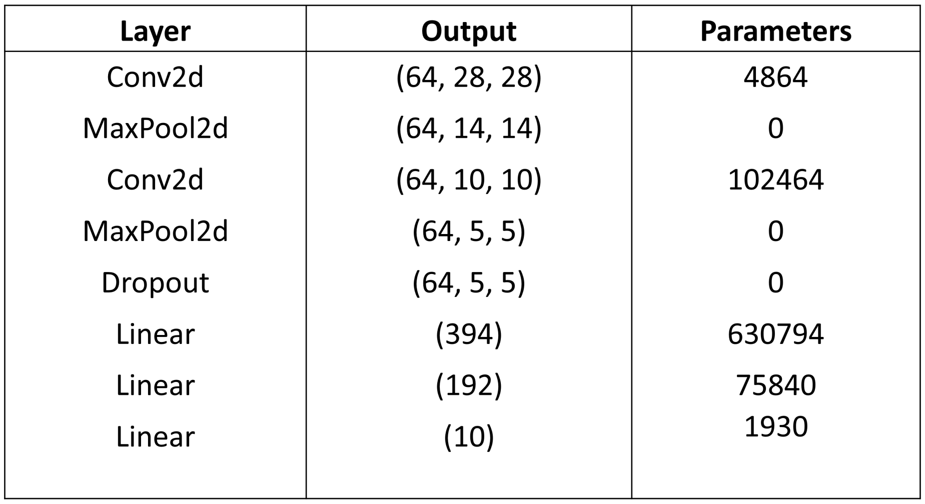
receive data from the overall distribution. Below is the pseudo code for data partitioning:

```
def data_partition(dataset, edge_clients):
{
    data_per_client = int(len(dataset)/edge_clients)
    clients = {}
    image = [i for i in range(len(dataset))]
    for i in range(edge_clients):
        clients[i] = set(np.random.choice(image, data_per_client))
        image = list(set(image) - clients[i])

    return clients
}
```

5.2 Experiments and Results Analysis

In this section, we analyse the performance of our proposed algorithm, Elastic Federated Learning (FedEN) in the presence of stragglers in edge federated learning. Further, we verify the effectiveness of FedEN in the case of statistical



Layer	Output	Parameters
Conv2d	(64, 28, 28)	4864
MaxPool2d	(64, 14, 14)	0
Conv2d	(64, 10, 10)	102464
MaxPool2d	(64, 5, 5)	0
Dropout	(64, 5, 5)	0
Linear	(394)	630794
Linear	(192)	75840
Linear	(10)	1930

Figure 5.4: CIFAR10 CNN

heterogeneity imposed due to partial results submission of heterogeneous edge devices, otherwise known as stragglers. We also discuss the effect of mixing parameter α in the local objectives of the participating edge devices. We then compare FedEN with the benchmark algorithms, FedAvg and FedProx. For simplicity, we consider federated training at one edge layer. The experiments are implemented with a well-known deep learning framework, PyTorch [110] with Python modules. The set up is described as follows:

System Heterogeneity: We set the percentage of stragglers as 10% and 50% of the edge devices during federated training, as a realistic scenario. This means that 10%/50% of edge devices do not perform all training rounds Therefore, 10% of computational heterogeneity (alias to system heterogeneity), mean the straggling devices are 10% out of total participating clients. Similarly, a 50 percentage of heterogeneity indicate that half of the participating edge devices are stragglers and the variance in data will be high causing divergence from the local objective.

Statistical Heterogeneity: Our primary objective is to perform image classification using the MNIST and CIFAR-10 datasets. MNIST consists of 60,000 training samples and 10,000 testing samples, whereas CIFAR10 has 50,000 training samples and 10,000 testing samples. We assume the edge devices have half of the data from a similar distribution. Hence, only half of the data of all the devices is not from the overall distribution. Therefore, we divide the data between the edge devices in such a way that each device has some data from the same distributions. In our case, each device will have some data from each of the 10 classes from the MNIST and CIFAR10 data set. Further, we consider two sources of statistical heterogeneity contributing to emergence of stragglers. The incomplete results or partial results submission causes the edge devices to diverge from the local objectives which enhances the straggling effect, thereby contributing to statistical heterogeneity. Similarly, the high dimensional data from the edge devices have a large number of parameters which further adds to data variance (statistical heterogeneity). Hence, the specified heterogeneity sources causes stragglers.

Training Hyper-parameters: For training on image classification using MNIST and CIFAR-10, we use convolutional neural networks. The hyper-parameter used is as follows:

Parameter	value
local epochs	10
local batch size	10
communication rounds	100
edge devices	100
client selection fraction (C)	0.01
learning rate	0.01

Table 5.3: Data sets

5.2.1 Implementation Details:

In this section, we describe in detail the implementation of our experiments along with some additional experiment results. The proposed algorithm is implemented using pytorch. Different modules are implemented for federated training such as local training, data aggregation at the server, random epoch generation based on the percentage of heterogeneity assumed in the system such as 10% and 50% of stragglers in the system.

We start with the pseudo code used for each of the modules used for implementing federated learning. **Data Partitioning:** MNIST and CIFAR-10 datasets are used for model training. Both datasets are publicly available. In real world, there will be no data sharing as data remains at the source where it is generated and only model parameters downloaded from the server are updated during local training and are returned to the server. In data partitioning module, data is shuffled and split between the selected clients. This shuffled data is then returned as a dictionary of indexed images to each of the client.

Models used in training: A simple convolutional neural network with two convolutions (one of them consisting of 32 channels and the other with 64 channels) are used. Next, a max pooling layer followed by fully connected layer consisting of 512 units. The activation function used is ReLu and the final output layer consists of softmax layer.

Local epochs generation: Based on the percentage of heterogeneity in a training round, the number of epochs required can be generated for selected clients. For instance, 'A' number of epochs are 10% of straggling devices. when the heterogeneity in the system is assumed as 10% then 10% of the clients or devices send their partial trained model results to the server and when the heterogeneity is 50% then 50% of the clients send their partial training results contributing in the global model. If there is no heterogeneity in the system, that is the number of stragglers in the system are zero which is an unrealistic situation in the present world due to the availability of millions of smart devices, then the specified number of epochs are run by the participating clients. In the other case, where the heterogeneity is present in the system and then the number of epoch performed by each client differs and are not the same.

Local Training at the Client Side: The weights of the global model are used with added elastic penalization in the local loss function of each client. Therefore, loss is calculated with the presence of elastic term. Different parameters are used by local training module such as the samples dataset, learning rate, number of epochs set for the local training. The optimizer used for model training is stochastic gradient descent. For a given set of local epochs, model parameters are updated and updated parameters are then returned to the server as a dictionary and total loss for that round of iteration. More local training can be performed by the clients to avoid frequent communications with the server thereby reducing the delays associated with server communication. Generally, when more local explorations are performed by the participating client in federated learning, it can cause the gradient to diverge from the local optima. However, this can be culminated by using elastic penalty which helps in making the objective function smooth and causing the parameters to shrink keeping all the parameters as close as possible to the local optima. **Server side Training:** After certain number of epochs for each training round by the participating clients, the server received the updated model parameters dictionary which then is used for model weights aggregation which is the average of all the received weights. Once, all the weights are averaged then the model is returned to the selected clients for further training rounds.

PSEUDO CODE FOR LOCAL TRAINING: Here is the pseudo code for local edge device training:

```
    class LocalTraining():
    def __init__(self, dataset, batch, lr, e , id, mu, alg):
    self.training_loader = DataLoader()
    self.lr = lr
    self.e = e

  def training(self, model_type):
    criterion = nn.CrossEntropyLoss()
    p_criterion = nn.MSELoss(reduction='mean')
    opt = torch.optim.SGD(model.parameters(), lr=self.lr, momentum=0.51)
```

```
Global_m = copy.deepcopy(Model_received)


start_t = time.time()


e_loss = []
for e in range(1, self.e+1):

  training_loss = 0.0

  Model.training()
  for data, labels in self.training_loader:

    if torch.cuda.is_available():
      data, label = data.cuda(), label.cuda()
      opt.zero_grad()
      out = model(data)

    _, pred = torch.max(out, 1)

    if self.algorithm <> 'feden':
      elastic term = 0.0

      # iterate through the current and global model parameters
      for w, w_t in zip(Model.parameters(), Global_m.parameters()) :

      loss = criterion(output, labels) + (elastic term)

    else:
      loss = criterion(output, labels)

    loss.backward()
    # optimization
    opt.step()
    # loss
    training_loss += loss.item()*data.size(0)

  # average of loss
  training_loss = training_loss/len(self.training_loader.dataset)
  A_loss.append(train_loss)

total_loss = sum(A_loss)/len(A_loss)

return Model.state_dict(), total_loss, (time.time() - start_time)
```

PSEUDO CODE FOR SERVER TRAINING

```
 def server_training(Model, r, batch, lr, ds, data, test_data, K, E, C
     , mu, pe, test_accuracy):

# global model weights
Global_wght = Model.state_dict()

# training loss
training_loss = []

# test accuracy
testing_acc = []

# store last loss for convergence
final_loss = 0.0

# total time taken
tot_time = 0

for curr_round in range(1, rounds+1):
  W, loc_loss, local_train_time = [], [], []

  M = max(int(C*K), 1)

  hetero_epochs = LocalEpochs(p, size=M, max_e=E)
  hetero_epochs = np.array(hetero_epochs)

  Str = np.random.choice(range(K), M, replace=False)
  Str = np.array(Str)

  # For Federated Averaging, stragglers are dropped
  if algorithm == 'fedavg':
    straggler = np.argwhere(hetero_epochs < E)
    hetero_epochs = np.delete(hetero_epochs, straggler)
    Str = np.delete(Str, straggler)

  for k, epoch in zip(S_t, hetero_epochs):
    local_updates =Updates(dataset=ds, batch=batch, lr=lr, e=e, id=
         data_dict[k], mu=mu, algo=algor)
    Weights, loss, local_training_time = local_updates.training(Model=
         copy.deepcopy(Model))

    w.append(copy.deepcopy(Weights))
    loss.append(copy.deepcopy(loss))
    train_time.append(train_time)
  # global weights time
  Global_time = time.time()

  # global weights updating
  Weights_avg = copy.deepcopy(W[0])
  for k in weights_avg.keys():
    for i in range(1, len(W)):
      weights_avg[k] += W[i][k]

    Weights_avg[k] = torch.div(Weights_avg[k], len(W))

  Global_weights = weights_avg

  Global_end_time = time.time()

  # calculate time
  total_time +=
  (Global_end_t - Global_start_t) + sum(local_train_time)/len(
      local_train_time)

  # move the updated weights to our model state dict
  model.load_state_dict(Global_weights)

  # loss
  Global_model= copy.deepcopy(Model)
  if algorithm == 'fedEN':
      l_avg = (sum(l_loss) / len(l_loss))


  # test accuracy
  criterion = nn.CrossEntropyLoss()
  testing_loss, testing_accuracy
  =  testing(Model, test_data, 128, criterion, number_classes,
      classes_test)



  training_loss.append(l_avg)

  test_accuracy.append(test_accuracy)
  if test_accuracy >= test_accuracy:
    rounds = current_train_round
    break


  # update the last loss
  last_loss = loss_avg

fig, ax = plt.subplots()
x_axis = np.arange(1, r+1)
```

```
  y_axis = np.array(training_loss)
  ax.plot(x_axis, y_axis)


  fig1, ax1 = plt.subplots()
  x_axis0 = np.arange(1, r+1)
  y_axis0 = np.array(test_accuracy)
  ax1.plot(x_axis0, y_axis0)


  return model
```

Network Architecture:

Various ML models can be used to address to the problem statement and requirements. Different models include linear regression, logistic regression, support vector machines (SVM), neural networks (NN), and Bayesian regression. Any of the aforementioned models could be used to find a solution to the problem. As we aim for image classification using the MNIST and CIFAR-10 datasets, we use convolutional neural networks. Convolutional neural networks are used for training on the image classification tasks [194, 199, 207]. Training is performed using the MNIST and CIFAR-10 image datasets. For MNIST, local training is performed on the training set. The MNIST model network starts with the input layer followed by the two convolutional layers of size 5x5. These convolutional layers are used for feature extraction and the linear layer at the end of the network acts as a classifier. Next, the convolutional layer is followed by max pooling of size 2x2. The max pooling layer helps in dimensionality reductions. The max pooling layer is followed by a fully connected layer of 512 units. The activation function used is ReLu. Finally, the output layer consists of softmax function for predicting the classification probabilities. For CIFAR-10, the CNN model has 64 5x5 filters for two convolutional layers. Next, the convolutional layers are followed by two fully connected layers. The first fully connected layer consists of 394 neurons and the second fully connected layer consists of 192 neurons. We use stochastic gradient descent (SGD) as the optimizer and a learning rate of 0.01. We adopt a local mini-batch size of 10. The CNN structure specified earlier is similar to the one specified by [96]. Further, baseline algorithms (FedAvg and FedProx) also use the same CNN architecture for a fair comparison. The CNN model architecture for both the MNIST and CIFAR-10 datasets is shown in Tables 5.4, 5.3.

Data Import: Following is the pseudocode for data import in Pytorch:

```
mnist_train = transforms.Compose([
                                    transforms.Resize((32, 32)),
                                    transforms.RandomCrop((28, 28)),
                                    transforms.RandomRotation(degrees=30),
                                    transforms.ToTensor(),\\
                                    transforms.Normalize((0.5,), (0.5,)),
                                    ])


mnist_test = transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Resize((32, 32)),
                                    transforms.RandomCrop((28, 28)),
                                    transforms.Normalize((0.5,), (0.5,)),
                                    ])


mnist_train = datasets.MNIST('./data/mnist/', train=True, download=True,
                            transform=mnist_train)
mnist_test = datasets.MNIST('../data/mnist/', train=False, download=True,
                            transform=mnist_test)




cifar10_train = transforms.Compose([
                                    #transforms.Grayscale(num_output_channels=1),
                                    #transforms.Resize((32, 32)),
                                    #transforms.RandomCrop((28, 28)),
                                    #transforms.RandomRotation(degrees=30),
                                    #transforms.GaussianBlur(kernel_size=501),
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5)),
                                    ])

cifar10_test = transforms.Compose([
                                    transforms.ToTensor(),
                                    #transforms.Resize((32, 32)),
                                    #transforms.RandomCrop((28, 28)),
                                    #transforms.Grayscale(num_output_channels=1),
                                    #transforms.GaussianBlur(kernel_size=501),
                                    transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5)),
                                    ])
```

Data Partitioning

```
    for i in range(edge_clients):
       random = set(np.random.choice(allocation_id, data_per_device, replace=False))
       allocation_id = list(set(allocation_id) - random)
    for r in random:
     edge_device_arr[i] = np.concatenate((edge_device_arr[i],
                         idxs[r*data_size:(r+1)*data_size]), axis=0)

  return edge_device_arr
```

Mnist model

```
class mnist(nn.module):
def __init__(self):
    super(mnist, self).__init__()

    self.conv1 = nn.conv2d(1, 32, kernel_Size=5)
    self.conv2 = nn.conv2d(32, 64, kernel_Size=5)

    self.pool = nn.maxpool2d(2,2)
    self.dropout = nn.dropout(0.2)

    self.fullyconnected1=nn.linear(1024,512)
    self.output=nn.Linear(512,10)

    def Forward(self, x):
    x = self.poo;(F.RELU(self.conv1(x)))
    x=self.pool()
    x=self.dropout()
    x= torch.flatten()
    x= F.RELU()
    x=self.output(x)
    output= F.log_softmax(x,1)

    Return output
```

Cifar-10 model

```
    class cifar10(nn.Module):

      def __init__(self):
        super(cifar10, self).__init__()

        self.conv1 = nn.conv2d(3, 64, kernel_size=5)
        self.conv2 = nn.conv2d(64, 64, kernel_size=5)

        self.pool = nn.Maxpool2d(2,2)
        self.dropout = nn.dropout(p=0.2)
        self.fullyconnectedc1 = nn.Linear(1600, 394)
        self.fullyconnectedc2 = nn.Linear(394, 192)
        self.out = nn.Linear(192,10)

      def Forward(self, x):
        x = self.pool(F.RELU(self.conv1(x)))
        x = self.pool(F.RELU(self.conv2(x)))
        x = self.Dropout(x)
        x = torch.Flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.Output(x)
        Output = F.log_softmax(x,1)

        return out
```

local epochs generation based on the edge devices heterogeneity aspect

5.2.2 Results Evaluation

The federated learning algorithms which we use as the benchmarks to compare our proposed model are as follows:

- FedAvg: Local update rule involves simple stochastic gradient descent for a number of iterations before communicating the results to the server.
- FedProx : Edge devices update their local model with a proximal term in their objective function. Here, we consider it as a benchmark since it also deals with statistical heterogeneity via a proximal term.

We compare the performance of our proposed model, FedEN, with these benchmarks in terms of the target accuracy reached over the number of communication rounds. We consider the partial device participation scenario where the edge devices contribute their partial results to the edge server since it cannot complete all training iterations. These devices are known as stragglers with different computation capabilities. Hence, data heterogeneity also spikes due to this type of participation. We use MNIST (consisting of 60,000 training and 10,000 testing samples) and CIFAR-10 (consisting of 50,000 training and 10,000 testing samples) for the classification task. We implement our algorithm in Python using PyTorch consisting of different modules for local training, and server training. Further, we incorporate 10 % and 50 % of system heterogeneity to replicate the real-world uncertainty of devices dropping out due to system heterogeneity.

h

Methods	R50	R100	R150	R200
FedAvg [96]	64.44	68.99	75.02	83.91
FedProx [81]	67.80	75.01	78.99	86.9
FedEN	71.94	78.9	85.5	**88.7**

Table 5.4: CIFAR10 Test Accuracy : 10% Stragglers

Methods	R50	R100	R150	R200
FedAvg	62.99	65.09	71.29	80.02
FedProx	65.80	73.99	76.02	83.50
FedEN	69.94	82.52	83.89	**86.95**

Table 5.5: CIFAR10 Test Accuracy : 50% Stragglers

Methods	R50	R100	R150	R200
FedAvg	88.02	90.02	93.01	95.01
FedProx	89.9	91.5	94.5	96.09
FedEN	92.7	94.89	96.83	**98.89**

Table 5.6: MNIST Test Accuracy : 10% Stragglers

Methods	R50	R100	R150	R200
FedAvg	87.21	89.5	92.15	94.3
FedProx	88.57	89.55	92.41	95
FedEN	91.81	92.89	95.98	**97.05**

Table 5.7: MNIST Test Accuracy : 50% Stragglers

Methods	R50	R100	R150	R200
FedAvg	0.048	0.043	0.043	0.040
FedProx	0.048	0.041	0.031	0.036
FedEN	0.046	0.038	0.037	**0.034**

Table 5.8: CIFAR-10 Train Loss: 10 % Stragglers

Methods	R50	R100	R150	R200
FedAvg	0.051	0.045	0.040	0.042
FedProx	0.050	0.043	0.035	0.038
FedEN	0.046	0.040	0.033	**0.035**

Table 5.9: CIFAR-10 Train Loss : 50 % Stragglers

Methods	R50	R100	R150	R200
FedAvg	0.039	0.029	0.028	0.025
FedProx	0.039	0.028	0.025	0.024
FedEN	0.036	0.027	0.025	**0.022**

Table 5.10: MNIST Train Loss: 10 % Stragglers

Methods	R50	R100	R150	R200
FedAvg	0.042	0.040	0.032	0.030
FedProx	0.039	0.035	0.029	0.028
FedEN	0.038	0.030	0.029	**0.025**

Table 5.11: MNIST Train Loss: 50 % Stragglers

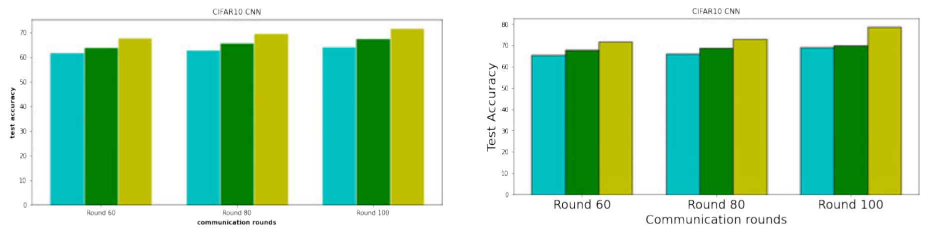


Figure 5.5: CIFAR10 Test Accuracy With 10% Stragglers

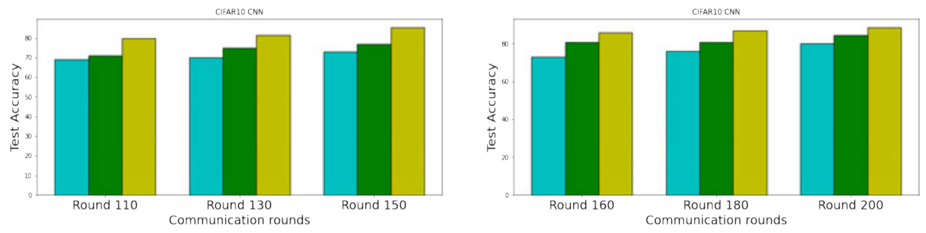


Figure 5.6: CIFAR10 Test Accuracy With 10% Stragglers

5.2.3 Performance comparison

CIFAR-10 - Test accuracy comparison for 10% stragglers: Table 5.4 shows the test results for accuracy for the CIFAR-10 dataset with 10% of stragglers. The accuracy over the test data samples is demonstrated over 50, 100, 150 and 200 communication rounds. For communication round 50, the test accuracy for FedAvg is 64.44 and for FedProx it is 67.80. However, FedEN achieves accuracy of 71.94 for 50 communication rounds. Similarly, for round 100, FedProx achieves an accuracy of 75.01 compared with FedAvg which has an accuracy of 68.99. However, FedEN achieves an accuracy of 78.9, which is higher than both FedAvg and FedProx. For communication round 150, FedEN again achieves the highest test accuracy of 85.5 compared to FedAvg and FedProx at 75.02 and 78.99, respectively. A comparison of all previous training rounds shows that FedEN always achieves better accuracy than the benchmark algorithms. For communication round 200, the accuracy for FedEN is 88.70, compared to FedAvg 83.91 and FedProx 86.9. Therefore, a comparison of all communication rounds shows that FedEN achieves the highest test accuracy for CIFAR-10 classification with 10% stragglers contributing to statistical heterogeneity.

CIFAR-10 - Test accuracy comparison for 50% stragglers Table 5.5 shows the test accuracy for the CIFAR-10 dataset with 50% stragglers. The accuracy over the test data samples is demonstrated over 50, 100, 150 and 200 communication rounds. For communication round 50, the test accuracy for FedAvg is 62.99 and for FedProx, it is 65.80. However, FedEN achieves accuracy of 69.94 for around 50 communication rounds. Similarly, for round 100, FedProx achieves an accuracy of 73.99 compared with FedAvg which has an accuracy of

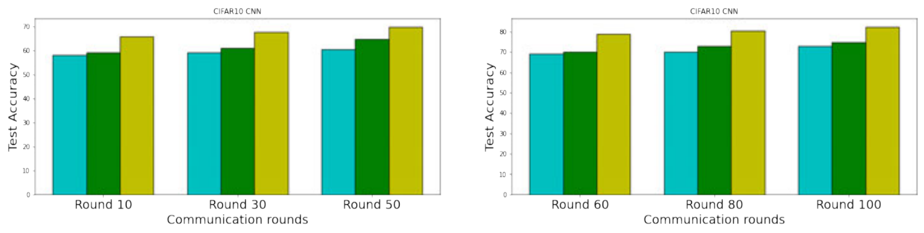


Figure 5.7: CIFAR10 Test Accuracy With 50% Stragglers

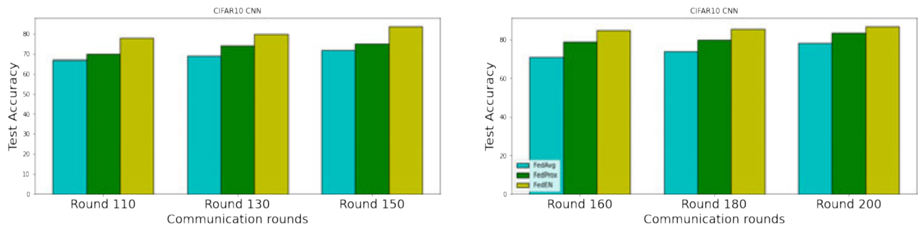


Figure 5.8: CIFAR10 Test Accuracy With 50% Stragglers

65.09. However, FedEN achieves an accuracy of 82.52, which is higher than FedAvg and FedProx. For communication round 150, FedEN achieves the highest test accuracy of 83.89 compared to FedAvg and FedProx at 71.29 and 76.02, respectively. A comparison of all previous training rounds shows that FedEN always achieves better accuracy than the benchmark algorithms. For communication round 200, FedEN achieves an accuracy of 86.95, compared to FedAvg at 80.02 and FedProx at 83.50. Therefore, a comparison of all the communication rounds shows that FedEN achieves the highest test accuracy for CIFAR-10 classification with 50% stragglers contributing to statistical heterogeneity causing variances in local data.

MNIST - Test accuracy comparison for 10% stragglers: Table 5.6 shows the , test results for accuracy for the MNIST dataset with 10% of stragglers. The accuracy over the test data samples is demonstrated over 50, 100, 150 and 200 communication rounds. For communication round 50, the test accuracy for FedAvg is 88.02 and for FedProx it is 89.9. However, FedEN achieves accuracy of 97.2 for around 50 communication rounds. For communication round 100, FedProx achieves an accuracy of 91.5 compared with FedAvg at 90.02. However, FedEN again achieves the highest accuracy of 94.89, compared to FedAvg and FedProx. For communication round 150, FedEN again achieves the highest test accuracy of 96.83 compared to FedAvg and FedProx, at 93.01 and 94.5, respectively. For communication round 200, the accuracy for FedAvg is 89.01 whereas FedProx achieves an accuracy of 96.09. FedEN achieves an accuracy of 98.89. Therefore, a comparison of all the communication rounds shows that, FedEN achieves the highest test accuracy for MNIST classification with

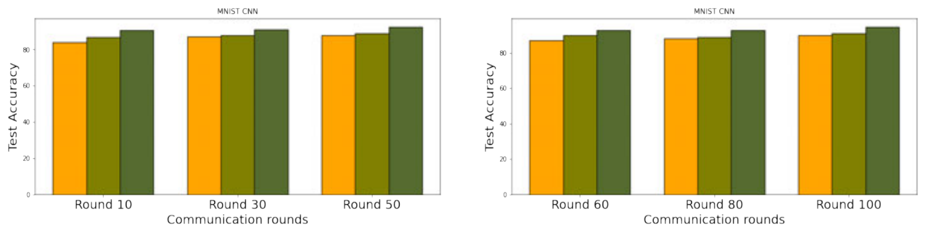


Figure 5.9: MNIST Test Accuracy With 10% Stragglers

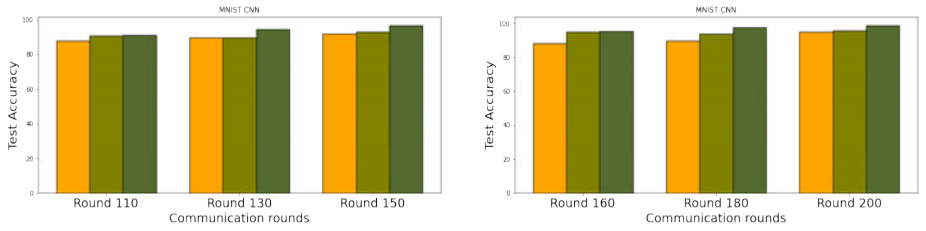


Figure 5.10: MNIST Test Accuracy With 10% Stragglers

10% stragglers contributing to statistical heterogeneity.

MNIST - Test accuracy comparison for 50% stragglers: Table 5.6 shows the , test results for accuracy for the MNIST dataset with 50% of stragglers. The accuracy over the test data samples is demonstrated over 50, 100, 150 and 200 communication rounds. For communication round 50, the test accuracy for FedAvg is 87.21 and for FedProx it is 88.57. However, FedEN achieves an accuracy of 91.81 for 50 communication rounds. For round 100, FedProx achieves an accuracy of 89.55 compared with FedAvg which has an accuracy of is 89.50. However, FedEN achieves an accuracy of 92.89, which is higher than both FedAvg and FedProx. For communication round 150, FedEN again achieves the highest test accuracy of 95.98 compared to FedAvg and FedProx at 92.15 and 92.41 respectively. For communication round 200, the accuracy for FedAvg is 94.3 compared to FedProx which achieves an accuracy of 95 and . FedEN which achieves an accuracy of 97.05. Therefore, a comparison comparing with all commu- nication rounds shows that, FedEN achieves the highest test accuracy for MNIST classification with 50% stragglers contributing to statistical heterogeneity.

Experiment conclusions: The graphical representation of the experiments dis- cussed earlier shows how the performance of federated training can be improved in the presence of stragglers.

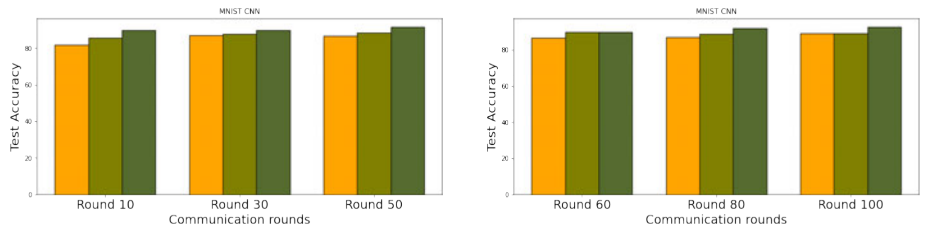


Figure 5.11: CIFAR10 Test Accuracy With 50% Stragglers

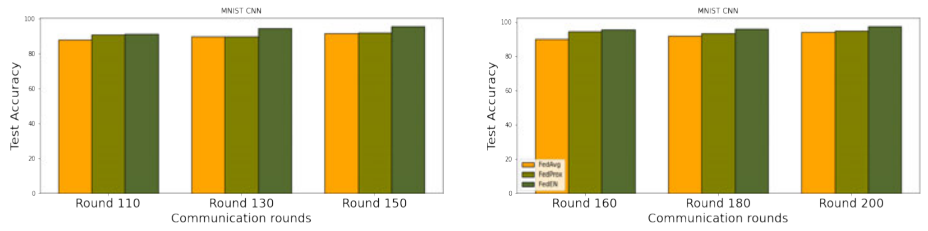


Figure 5.12: MNIST Test Accuracy With 50% Stragglers

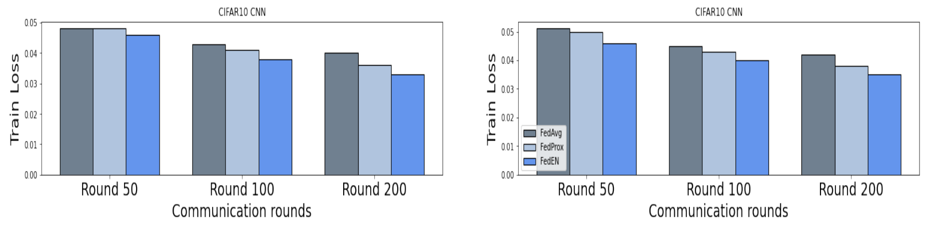


Figure 5.13: CIFAR10 Train Loss with 10% & 50% Stragglers

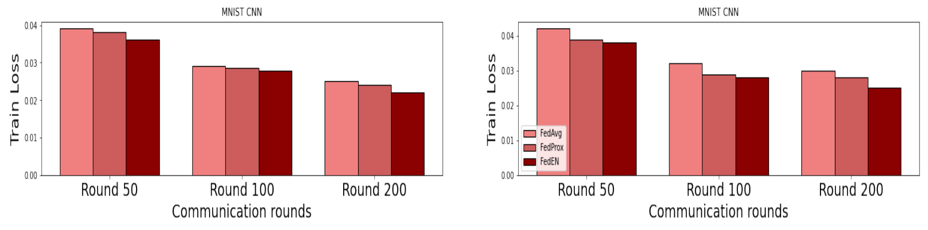


Figure 5.14: Mnist Train Loss With 10% & 50% Stragglers

Chapter 6

Conclusions

Overview

The conclusion to this chapter details how the research questions have been answered throughout the thesis.

6.1 Discussions

From the conclusions in chapter 3 and 4, the straggler problem is identified and an elastic optimized algorithm is proposed for the edge federated learning setting. In chapter 4, the theoretical framework for stragglers in edge federated learning is defined where we formulated stragglers as a distributed optimization problem. The setting mentioned above is a more realistic form of federated learning where there are millions of edge devices and each device has its own set of challenges in terms of participation. But, only the devices which satifsy the following conditions can participate in federated learning:

- Devices must be connected to a wireless network (such as Wifi)
- Devices must not perform any other activities while training
- Devices must be on charge while performing training

The proposed algorithm, FedEN, has the following advantages over the benchmarks in edge federated learning:

- FedEN allows data from stragglers to form the collaborative model where we set the percentage of stragglers as 10% and 50% of the edge devices during federated training which means that 10%/50% of edge devices do not perform all training rounds.
- FedEN reduces the number of communication rounds to reach target accuracy compared to the benchmark algorithms

- A balance between a reduction in the number of parameters and their shrinkage makes the objective function smooth and improves prediction accuracy.

As a consequence, it can be seen that the proposed algorithm for the proposed setting of edge federated learning can achieve a federated trained collaborative model which can better handle the inference of test data thereby achieving improved accuracy compared to the benchmark algorithms in the literature, such as FedAvg and FedProx. Based on the experiment results using the MNIST and Cifar-10 datasets detailed in chapter 5, it can be concluded that the performance not only depends on the percentage of stragglers present but also on the mixing parameter in the local objectives of edge devices. Numerical experiments show that the fewer stragglers there are in the training process, the better the accuracy of the collaborative predictive model. However, we cannot always assume that the system does not have any stragglers nor should we ignore them completely. We can instead make a trade-off between accuracy and the percentage of stragglers in federated training. The elastic mixing parameter controls the amount of lasso and ridge which is imposed in the local objection loss function. The mixing parameter can range between 0and1 depending on which penalization is imposed more in minimizing the local loss function.

6.2 Conclusion

The aim of our research was to deal with stragglers in edge federated learning with elastic optimization in the local objectives of the edge devices and to evaluate its performance against benchmark algorithms. Two hypotheses motivated our research. Our first hypothesis is that stragglers emerge due to high dimensional data set parameters which are millions in number. These parameters cause devices to perform delayed training as the device has to train all the parameters. However, the L_1 part of the elastic net helps in better feature selection and reduces the redundant parameters from training and improves training efficiency. We consider $\lambda = 1$ as the tuning parameter. Our second hypothesis the computational in capabilities of participating edge devices which can contribute to stragglers.Therefore, the L_2 part of the elastic net adds smoothness to the objective function by shrinking the parameters so that the local optimal can be found. Our experiments indicate that having elastic optimization in the local objectives of the participating edge devices can improve training performance by reducing the number of communication rounds required to achieve target accuracy. Our proposed algorithm, Elastic Optimized Federated Learning (FedEN), performs better than benchmark algorithms such as FedAvg and FedProx by achieving better accuracy and less training loss. FedEN is the reparameterization of FedAvg and FedProx such that the tuning parameter λ and the mixing parameter can alter the training performance. When $\lambda = 0$, FedEN is similar to FedAvg. Furthermore, when the mixing parameter $\alpha = 0$, then FedEN is similar to FedProx where the proximal term is used in the local objectives. Stragglers which emerge due to large number of parameters from high dimensional data from IoT edge devices causes a computation burden on edge devices. Similarly, when 10% and 50% of edge devices contribute partial results, the statistical heterogeneity worsens thereby causing gradient divergence.However, FedEN can balance between the two penalization mentioned earlier which is lasso and ridge, thereby producing sparse models with better prediction accuracy. In terms of test accuracy, our algorithm achieves better accuracy of 88.7 over 200 communication rounds and 10% of stragglers. Further, our algorithm achieves an accuracy of 86.95 with 50% of stragglers. Compared to benchmark algorithms such as FedAvg and FedProx, our algorithm achieves better accuracy over fewer communication rounds. Therefore, there are many possible research directions that can be investigated for straggler mitigation. For instance, each edge device data distribution can be analyzed using Bayesian inference so that devices with the same distribution can be segregated into one category and then federated learning can be performed, which helps in better edge device selection, thereby preventing stragglers caused by non-iid data distribution [112, 72, 60].Hence, by identifying the data distributions, edge devices can be allocated to a domain with the same distribution, thereby more accurate prediction models can be obtained. Finally, the verdict of optimizing training performance due to stragglers in edge federated learning is that in a real-time and dynamic setting, the observations made in the proposed algorithm can improve training performance while improving test accuracy.

Limitations of FedEN: The main limitation of FedEN is the uncertainty about the actual percentage of stragglers present in the training. Additionally, the ad-hoc nature of wireless connectivity as well as system heterogeneity can cause an edge device which is working perfectly at first to become a straggler at any point in the training time or during any iteration round. We have only investigated model training with convolutional neural networks. Training on different neural networks has not been investigated which would open doors to address many training challenges for edge devices. In addition to this, cross-validation for tuning parameter could possibly be an option to further improve the training and testing results of the proposed algorithm.

6.3 Future work

In future work, the existing studies related to edge device selection and integration with Bayesian inference can be further investigated which can improve the process of distribution identification among the whole population of edge devices. Dynamic algorithms can be integrated with FedEN which can make decisions on the go depending on how the edge devices perform training in each round and can categorize the devices of least priority in the tier of training participation.

Bibliography

[1] Alaa Awad Abdellatif, Naram Mhaisen, Amr Mohamed, Aiman Erbad, Mohsen Guizani, Zaher Dawy, and Wassim Nasreddine. Communication-efficient hierarchical federated learning for iot heterogeneous systems with imbalanced data. *Future Generation Computer Systems*, 128:406–419, 2022.

[2] Ahmed M Abdelmoniem and Marco Canini. Towards mitigating device heterogeneity in federated learning via adaptive model quantization. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 96–103, 2021.

[3] Md Momin Al Aziz, Md Monowar Anjum, Noman Mohammed, and Xiaoqian Jiang. Generalized genomic data sharing for differentially private federated learning. *Journal of Biomedical Informatics*, page 104113, 2022.

[4] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Communications Surveys & Tutorials*, 22(3):1646–1685, 2020.

[5] Haider Al-Lawati and Stark C Draper. Gradient staleness in asynchronous optimization under random communication delays. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4353–4357. IEEE, 2022.

[6] Jonatha Anselmi and Neil Walton. Stability and optimization of speculative queueing networks. *IEEE/ACM Transactions on Networking*, 30(2):911–922, 2021.

[7] Ajay Badita, Parimal Parag, and Vaneet Aggarwal. Single-forking of coded subtasks for straggler mitigation. *IEEE/ACM Transactions on Networking*, 29(6):2413–2424, 2021.

[8] Kamalakant Laxman Bawankule, Rupesh Kumar Dewang, and Anil Kumar Singh. Early straggler tasks detection by recurrent neural network in a heterogeneous environment. *Applied Intelligence*, pages 1–21, 2022.

[9] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[10] Rawad Bitar, Mary Wootters, and Salim El Rouayheb. Stochastic gradient coding for straggler mitigation in distributed learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):277–291, 2020.

[11] Baturalp Buyukates, Emre Ozfatura, Sennur Ulukus, and Deniz Gündüz. Gradient coding with dynamic clustering for straggler-tolerant distributed learning. *IEEE Transactions on Communications*, 2022.

[12] Dongqi Cai, Tao Fan, Yan Kang, Lixin Fan, Mengwei Xu, Shangguang Wang, and Qiang Yang. Accelerating vertical federated learning. *IEEE Transactions on Big Data*, pages 1–10, 2022.

[13] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 125–136, 2020.

[14] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. *Advances in neural information processing systems*, 34:20461–20475, 2021.

[15] Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li. Accelerating distributed learning in non-dedicated environments. *IEEE Transactions on Cloud Computing*, 2021.

[16] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[17] Zunming Chen, Hongyan Cui, Ensen Wu, and Xi Yu. Dynamic asynchronous anti poisoning federated deep learning with blockchain-based reputation-aware solutions. *Sensors*, 22(2):684, 2022.

[18] Shekha Chenthara, Khandakar Ahmed, Hua Wang, Frank Whittaker, and Zhenxiang Chen. Healthchain: A novel framework on privacy preservation of electronic health records using blockchain technology. *Plos one*, 15(12):e0243043, 2020.

[19] Dalsu Choi, Hyunsik Yoon, and Yon Dohn Chung. Resky: Efficient subarray skyline computation in array databases. *Distributed and Parallel Databases*, pages 1–38, 2022.

[20] Thinh Quang Dinh, Diep N Nguyen, Dinh Thai Hoang, Tran Vu Pham, and Eryk Dutkiewicz. In-network computation for large-scale federated learning over wireless edge networks. *IEEE Transactions on Mobile Computing*, 2022.

[21] Kevin Dowd and Charles Severance. High performance computing. 2010.

[22] Sanghamitra Dutta, Jianyu Wang, and Gauri Joshi. Slow and stale gradients can win the race. *IEEE Journal on Selected Areas in Information Theory*, 2(3):1012–1024, 2021.

[23] Nuwan Ferdinand, Haider Al-Lawati, Stark C Draper, and Matthew Nokleby. Anytime minibatch: Exploiting stragglers in online distributed optimization. *arXiv preprint arXiv:2006.05752*, 2020.

[24] Yong-Feng Ge, Jinli Cao, Hua Wang, Zhenxiang Chen, and Yanchun Zhang. Set-based adaptive distributed differential evolution for anonymity-driven database fragmentation. *Data Science and Engineering*, pages 1–12, 2021.

[25] Yong-Feng Ge, Jinli Cao, Hua Wang, Jiao Yin, Wei-Jie Yu, Zhi-Hui Zhan, and Jun Zhang. A benefit-driven genetic algorithm for balancing privacy and utility in database fragmentation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 771–776, 2019.

[26] Yong-Feng Ge, Maria Orlowska, Jinli Cao, Hua Wang, and Yanchun Zhang. Knowledge transfer-based distributed differential evolution for dynamic database fragmentation. *Knowledge-Based Systems*, page 107325, 2021.

[27] Yong-Feng Ge, Maria Orlowska, Jinli Cao, Hua Wang, and Yanchun Zhang. Mdde: multitasking distributed differential evolution for privacy-preserving database fragmentation. *The VLDB Journal*, pages 1–19, 01 2022.

[28] Yong-Feng Ge, Wei-Jie Yu, Jinli Cao, Hua Wang, Zhi-Hui Zhan, Yanchun Zhang, and Jun Zhang. Distributed memetic algorithm for outsourced database fragmentation. *IEEE Transactions on Cybernetics*, 51(10):4808–4821, 2021.

[29] Xinran Gu, Kaixuan Huang, Jingzhao Zhang, and Longbo Huang. Fast federated learning in the presence of arbitrary device unavailability. *Advances in Neural Information Processing Systems*, 34:12052–12064, 2021.

[30] Jingjing Guo, Haiyang Li, Feiran Huang, Zhiquan Liu, Yanguo Peng, Xinghua Li, Jianfeng Ma, Varun G Menon, and Konstantin Igorevich Kostromitin. Adfl: A poisoning attack defense framework for horizontal federated learning. *IEEE Transactions on Industrial Informatics*, 2022.

[31] Manupriya Gupta, Pavas Goyal, Rohit Verma, Rajeev Shorey, and Huzur Saran. Fedfm: Towards a robust federated learning approach for fault mitigation at the edge nodes. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 362–370. IEEE, 2022.

[32] Rami Hamdi, Mingzhe Chen, Ahmed Ben Said, Marwa Qaraqe, and H Vincent Poor. Federated learning over energy harvesting wireless networks. *IEEE Internet of Things Journal*, 9(1):92–103, 2021.

[33] Ahmad Hammoud, Hadi Otrok, Azzam Mourad, and Zbigniew Dziong. On demand fog federations for horizontal federated learning in iov. *IEEE Transactions on Network and Service Management*, 2022.

[34] Serge Kas Hanna, Rawad Bitar, Parimal Parag, Venkat Dasari, and Salim El Rouayheb. Adaptive stochastic gradient descent for fast and communication-efficient distributed learning. *arXiv preprint arXiv:2208.03134*, 2022.

[35] Jing He, Songtao Guo, Mingyan Li, and Yongdong Zhu. Acefl: Federated learning accelerating in 6g-enabled mobile edge computing networks. *IEEE Transactions on Network Science and Engineering*, 2022.

[36] Jing He, Songtao Guo, Dewen Qiao, and Lin Yi. Hetefl: network-aware federated learning optimization in heterogenous mec-enabled internet of things. *IEEE Internet Things J*, 2022.

[37] Jinyuan He, Jia Rong, Le Sun, Hua Wang, Yanchun Zhang, and Jiangang Ma. *D-ECG: A Dynamic Framework for Cardiac Arrhythmia Detection from IoT-Based ECGs: 19th International Conference, Dubai, United Arab Emirates, November 12-15, 2018, Proceedings, Part II*, pages 85–99. 11 2018.

[38] Xiaofan He, Tianheng Li, Richeng Jin, and Huaiyu Dai. Delay-optimal coded offloading for distributed edge computing in fading environments. *IEEE Transactions on Wireless Communications*, 2022.

[39] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

[40] Seyyedali Hosseinalipour, Christopher G Brinton, Vaneet Aggarwal, Huaiyu Dai, and Mung Chiang. From federated to fog learning: Distributed machine learning over heterogeneous wireless networks. *IEEE Communications Magazine*, 58(12):41–47, 2020.

[41] Jiajia Huang, Min Peng, Hua Wang, Jinli Cao, Wang Gao, and Xiuzhen Zhang. A probabilistic method for emerging topic tracking in microblog stream. *World Wide Web*, 20(2):325–350, 2017.

[42] Shanfeng Huang, Zezhong Zhang, Shuai Wang, Rui Wang, and Kaibin Huang. Accelerating federated edge learning via topology optimization. *IEEE Internet of Things Journal*, pages 1–1, 2022.

[43] Shudong Huang, Wei Shi, Zenglin Xu, Ivor W Tsang, and Jiancheng Lv. Efficient federated multi-view learning. *Pattern Recognition*, page 108817, 2022.

[44] Ting Huang, Yue-Jiao Gong, Wei-Neng Chen, Hua Wang, and Jun Zhang. A probabilistic niching evolutionary computation framework based on binary space partitioning. *IEEE Transactions on Cybernetics*, 52(1):51–64, 2022.

[45] Ting Huang, Yue-Jiao Gong, Sam Kwong, Hua Wang, and Jun Zhang. A niching memetic algorithm for multi-solution traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 24(3):508–522, 2019.

[46] HamidReza Imani, Jeff Anderson, and Tarek El-Ghazawi. isample: Intelligent client sampling in federated learning. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 58–65. IEEE, 2022.

[47] Tayyebeh Jahani-Nezhad and Mohammad Ali Maddah-Ali. Berrut approximated coded computing: Straggler resistance beyond polynomial computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[48] Zeyu Ji, Xingjun Zhang, Jingbo Li, Jia Wei, and Zheng Wei. Ep4ddl: addressing straggler problem in heterogeneous distributed deep learning. *The Journal of Supercomputing*, pages 1–18, 2022.

[49] Zhongming Ji, Li Chen, Nan Zhao, Yunfei Chen, Guo Wei, and F. Richard Yu. Computation offloading for edge-assisted federated learning. *IEEE Transactions on Vehicular Technology*, 70(9):9330–9344, 2021.

[50] Jiayin Jin, Jiaxiang Ren, Yang Zhou, Lingjuan Lyu, Ji Liu, and Dejing Dou. Accelerated federated learning with decoupled adaptive optimization. In *International Conference on Machine Learning*, pages 10298–10322. PMLR, 2022.

[51] Enamul Kabir. A role-involved purpose-based access control model. *Information Systems Frontiers*, 14:809–822, 07 2012.

[52] Enamul Kabir, Jiankun Hu, Hua Wang, and Guangping Zhuo. A novel statistical technique for intrusion detection systems. *Future Generation Computer Systems*, 79:303–318, 2018.

[53] Enamul Kabir and Hua Wang. Conditional purpose based access control model for privacy protection. In *Proceedings of the Twentieth Australasian Conference on Australasian Database*, volume 92, pages 137–144, 01 2009.

[54] Md Enamul Kabir and Hua Wang. Microdata protection method through microaggregation: a systematic approach. *Journal of Software*, 7(11):2415–2422, 2012.

[55] Md Enamul Kabir, Hua Wang, and Elisa Bertino. A conditional purpose-based access control model with dynamic roles. *Expert Systems with Applications*, 38(3):1482–1489, 2011.

[56] Md Enamul Kabir, Hua Wang, and Elisa Bertino. Efficient systematic clustering method for k-anonymization. *Acta Informatica*, 48(1):51–66, 2011.

[57] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[58] Nurullah Karakoç, Anna Scaglione, Martin Reisslein, and Ruiyuan Wu. Federated edge network utility maximization for a multi-server system: algorithm and convergence. *IEEE/ACM Transactions on Networking*, 2022.

[59] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[60] Faten Khalil, Hua Wang, and Jiuyong Li. Integrating markov model with clustering for predicting web page accesses. In *Proceeding of the 13th Australasian world wide web conference*, pages 63–74, 01 2007.

[61] Latif U Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Communications Surveys & Tutorials*, 2021.

[62] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[63] Ivan Kholod, Evgeny Yanaki, Dmitry Fomichev, Evgeniy Shalugin, Evgenia Novikova, Evgeny Filippov, and Mats Nordlund. Open-source federated learning frameworks for iot: A comparative review and analysis. *Sensors*, 21(1):167, 2020.

[64] Ivan I Kholod, Mikhail A Efremov, Maxim A Kolpashikov, Andrey V Vasilyev, Pavel L Tabakov, and Ilya E Aristarhov. Fl4j—federated learning framework for java. In *International Symposium on Intelligent and Distributed Computing*, pages 225–234. Springer, 2022.

[65] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[66] Vasileios Kouliaridis, Georgios Kambourakis, Efstratios Chatzoglou, Dimitrios Geneiatakis, and Hua Wang. Dissecting contact tracing apps in the android platform. *Plos one*, 16(5):e0251867, 2021.

[67] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pages 11814–11827. PMLR, 2022.

[68] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pages 19–35, 2021.

[69] Hyun-Suk Lee and Jang-Won Lee. Adaptive transmission scheduling in wireless networks for asynchronous federated learning. *IEEE Journal on Selected Areas in Communications*, 39(12):3673–3687, 2021.

[70] Guanghao Li, Yue Hu, Miao Zhang, Ji Liu, Quanjun Yin, Yong Peng, and Dejing Dou. Fedhisyn: A hierarchical synchronous federated learning framework for resource and data heterogeneity. *arXiv preprint arXiv:2206.10546*, 2022.

[71] Hu Li, Ye Wang, Hua Wang, and Bin Zhou. Multi-window based ensemble learning for classification of imbalanced streaming data. *World Wide Web*, 20(6):1507–1525, 2017.

[72] Jian-Yu Li, Ke-Jing Du, Zhi-Hui Zhan, Hua Wang, and Jun Zhang. Distributed differential evolution with adaptive resource allocation. *IEEE Transactions on Cybernetics*, pages 1–14, 2022.

[73] Jian-Yu Li, Zhi-Hui Zhan, Hua Wang, and Jun Zhang. Data-driven evolutionary algorithm with perturbation-based ensemble surrogates. *IEEE Transactions on Cybernetics*, 51(8):3925–3937, 2021.

[74] Jie Li, Okko Makkonen, Camilla Hollanti, and Oliver W Gnilke. Efficient recovery of a shared secret via cooperation: Applications to sdmm and pir. *IEEE Journal on Selected Areas in Communications*, 40(3):871–884, 2022.

[75] Jun Li, Xiaoman Shen, Lei Chen, and Jiajia Chen. Bandwidth slicing to boost federated learning over passive optical networks. *IEEE Communications Letters*, 24(7):1492–1495, 2020.

[76] Li Li, Duo Liu, Moming Duan, Yu Zhang, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. Federated learning with workload-aware client scheduling in heterogeneous systems. *Neural Networks*, 2022.

[77] Lintao Li, Longwei Yang, Xin Guo, Yuanming Shi, Haiming Wang, Wei Chen, and Khaled B Letaief. Delay analysis of wireless federated learning based on saddle point approximation and large deviation theory. *IEEE Journal on Selected Areas in Communications*, 39(12):3772–3789, 2021.

[78] Min Li, Xiaoxun Sun, Hua Wang, and Yanchun Zhang. Multi-level delegations with trust management in access control systems. *Journal of Intelligent Information Systems*, 39(3):611–626, 2012.

[79] Min Li, Xiaoxun Sun, Hua Wang, Yanchun Zhang, and Ji Zhang. Privacy-aware access control with trust management in web service. *World Wide Web*, 14(4):407–430, 2011.

[80] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[81] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[82] Youqi Li, Fan Li, Lixing Chen, Liehuang Zhu, Pan Zhou, and Yu Wang. Power of redundancy: Surplus client scheduling for federated learning against user uncertainties. *IEEE Transactions on Mobile Computing*, 2022.

[83] Zhuojin Li, Marco Paolieri, Leana Golubchik, Sung-Han Lin, and Wumo Yan. Predicting throughput of distributed stochastic gradient descent. *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[84] Zonghang Li, Huaman Zhou, Tianyao Zhou, Hongfang Yu, Zenglin Xu, and Gang Sun. Esync: Accelerating intra-domain federated learning in heterogeneous data centers. *IEEE Transactions on Services Computing*, 2020.

[85] Zehong Lin, Hang Liu, and Ying-Jun Angela Zhang. Relay-assisted cooperative federated learning. *IEEE Transactions on Wireless Communications*, pages 1–1, 2022.

[86] Fan Liu, Xingshe Zhou, Jinli Cao, Zhu Wang, Tianben Wang, Hua Wang, and Yanchun Zhang. Anomaly detection in quasi-periodic time series based on automatic data segmentation and attentional lstm-cnn. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2626–2640, 2022.

[87] Jinwei Liu. Fregata: A low-latency and resource-efficient scheduling for heterogeneous jobs in clouds. In *2022 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 15–22. IEEE, 2022.

[88] Juncai Liu, Jessie Hui Wang, Chenghao Rong, Yuedong Xu, Tao Yu, and Jilong Wang. Fedpa: An adaptively partial model aggregation strategy in federated learning. *Computer Networks*, 199:108468, 2021.

[89] Lewis Liu and Kun Zhao. Asynchronous stochastic gradient descent for extreme-scale recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 328–335, 2021.

[90] Sheng Liu, Qiyang Chen, and Linlin You. Fed2a: Federated learning mechanism in asynchronous and adaptive modes. *Electronics*, 11(9):1393, 2022.

[91] Shengli Liu, Guanding Yu, Rui Yin, Jiantao Yuan, Lei Shen, and Chonghe Liu. Joint model pruning and device selection for communication-efficient federated edge learning. *IEEE Transactions on Communications*, 70(1):231–244, 2021.

[92] Wei-Li Liu, Yue-Jiao Gong, Wei-Neng Chen, Zhiqin Liu, Hua Wang, and Jun Zhang. Coordinated charging scheduling of electric vehicles: A mixed-variable differential evolution approach. *IEEE Transactions on Intelligent Transportation Systems*, 21(12):5094–5109, 2019.

[93] Wentao Liu, Xiaolong Xu, Lianxiang Wu, Lianyong Qi, Alireza Jolfaei, Weiping Ding, and Mohammad R Khosravi. Intrusion detection for maritime transportation systems with batch federated aggregation. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[94] Yingchi Mao, Jun Wu, Xiaoming He, Ping Ping, Jiajun Wang, and Jie Wu. Joint dynamic grouping and gradient coding for time-critical distributed machine learning in heterogeneous edge networks. *IEEE Internet of Things Journal*, 2022.

[95] Othmane Marfoq, Chuan Xu, Giovanni Neglia, and Richard Vidal. Throughput-optimal topology design for cross-silo federated learning. *Advances in Neural Information Processing Systems*, 33:19478–19487, 2020.

[96] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[97] Jed Mills, Jia Hu, and Geyong Min. Multi-task federated learning for personalised deep neural networks in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):630–641, 2021.

[98] Jed Mills, Jia Hu, and Geyong Min. Client-side optimisation strategies for communication-efficient federated learning. *IEEE Communications Magazine*, 2022.

[99] A Navia-Vázquez, R Díaz-Morales, and M Fernández-Díaz. Budget distributed support vector machine for non-id federated learning scenarios. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[100] Jer Shyuan Ng, Wei Yang Bryan Lim, Zehui Xiong, Xianbin Cao, Dusit Niyato, Cyril Leung, and Dong In Kim. A hierarchical incentive design toward motivating participation in coded federated learning. *IEEE Journal on Selected Areas in Communications*, 40(1):359–375, 2021.

[101] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 3581–3607. PMLR, 2022.

[102] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[103] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Guoliang Xing, and Jianwei Huang. Clusterfl: A clustering-based federated learning system for human activity recognition. *ACM Transactions on Sensor Networks (TOSN)*, 2022.

[104] Dinesh Pandey, Hua Wang, Xiaoxia Yin, Kate Wang, Yanchun Zhang, and Jing Shen. *Automatic Breast Lesion Segmentation Using Continuous Max-Flow Algorithm in Phase Preserved DCE-MRIs*, pages 124–137. 11 2021.

[105] Shashi Raj Pandey, Minh N. H. Nguyen, Tri Nguyen Dang, Nguyen H. Tran, Kyi Thar, Zhu Han, and Choong Seon Hong. Edge-assisted democratized learning toward federated analytics. *IEEE Internet of Things Journal*, 9(1):572–588, 2022.

[106] Jinlong Pang, Ziyi Han, Ruiting Zhou, Haisheng Tan, and Yue Cao. Online scheduling algorithms for unbiased distributed learning over wireless edge networks. *Journal of Systems Architecture*, page 102673, 2022.

[107] Jinlong Pang, Jieling Yu, Ruiting Zhou, and John CS Lui. An incentive auction for heterogeneous client selection in federated learning. *IEEE Transactions on Mobile Computing*, 2022.

[108] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. Sageflow: Robust federated learning against both stragglers and adversaries. *Advances in Neural Information Processing Systems*, 34:840–851, 2021.

[109] Juwon Park, Daegun Yoon, Sangho Yeo, and Sangyoon Oh. Amble: Adjusting mini-batch and local epoch for federated learning with heterogeneous devices. *Journal of Parallel and Distributed Computing*, 2022.

[110] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[111] Min Peng, Wang Gao, Hua Wang, Yanchun Zhang, Jiajia Huang, Qianqian Xie, Gang Hu, and Gang Tian. Parallelization of massive textstream compression based on compressed sensing. *ACM Transactions on Information Systems (TOIS)*, 36(2):1–18, 2017.

[112] Min Peng, Qianqian Xie, Hua Wang, Yanchun Zhang, and Gang Tian. Bayesian sparse topical coding. *IEEE Transactions on Knowledge and Data Engineering*, PP:1–1, 06 2018.

[113] Min Peng, Guanyin Zeng, Zhaoyu Sun, Jiajia Huang, Hua Wang, and Gang Tian. Personalized app recommendation based on app permissions. *World Wide Web*, 21(1):89–104, 2018.

[114] Pei Peng and Emina Soljanin. Covert, low-delay, coded message passing in mobile (iot) networks. *IEEE Transactions on Information Forensics and Security*, 17:599–611, 2022.

[115] Pei Peng, Emina Soljanin, and Philip Whiting. Diversity/parallelism trade-off in distributed systems with redundancy. *IEEE Transactions on Information Theory*, 68(2):1279–1295, 2021.

[116] Pinyarash Pinyoanuntapong, Wesley Houston Huff, Minwoo Lee, Chen Chen, and Pu Wang. Toward scalable and robust aiot via decentralized federated learning. *IEEE Internet of Things Magazine*, 5(1):30–35, 2022.

[117] Saurav Prakash, Sagar Dhakal, Mustafa Riza Akdeniz, Yair Yona, Shilpa Talwar, Salman Avestimehr, and Nageen Himayat. Coded computing for low-latency federated learning over wireless edge networks. *IEEE Journal on Selected Areas in Communications*, 39(1):233–250, 2020.

[118] Qiao Qi and Xiaoming Chen. Robust design of federated learning for edge-intelligent networks. *IEEE Transactions on Communications*, 2022.

[119] Guannan Qu and Na Li. Accelerated distributed nesterov gradient descent. *IEEE Transactions on Automatic Control*, 65(6):2566–2581, 2019.

[120] Zhe Qu, Rui Duan, Lixing Chen, Jie Xu, Zhuo Lu, and Yao Liu. Context-aware online client selection for hierarchical federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[121] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. Openfl: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021.

[122] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. Adaptive node participation for straggler-resilient federated learning. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8762–8766, 2022.

[123] Nuria Rodríguez-Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámara, Gerardo González-Seco, M Victoria Luzón, Miguel Angel Veganzones, and Francisco Herrera. Federated learning and differential privacy: Software tools analysis, the sherpa. ai fl framework and methodological guidelines for preserving data privacy. *Information Fusion*, 64:270–292, 2020.

[124] Rubina Sarki, Khandakar Ahmed, Hua Wang, and Yanchun Zhang. Automated detection of mild and multi-class diabetic eye diseases using deep learning. *Health Information Science and Systems*, 8, 10 2020.

[125] Tushar Semwal, Ajinkya Mulay, and Ayush Manish Agrawal. Fedperf: A practitioners' guide to performance of federated learning algorithms. 2020.

[126] Yulong Shen, Tao Zhang, Yongzhi Wang, Hua Wang, and Xiaohong Jiang. Microthings: A generic iot architecture for flexible data aggregation and scalable service cooperation. *IEEE Communications Magazine*, 55:86–93, 01 2017.

[127] Gaoxin Shi, Shuaishuai Guo, Jia Ye, Nasir Saeed, and Shuping Dang. Multiple parallel federated learning via over-the-air computation. *IEEE Open Journal of the Communications Society*, 2022.

[128] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[129] Wen Shi, Wei-Neng Chen, Sam Kwong, Jie Zhang, Hua Wang, Tianlong Gu, Huaqiang Yuan, and Jun Zhang. A coevolutionary estimation of distribution algorithm for group insurance portfolio. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (10.1109/TSMC.2021.3096013):1–15, 2021.

[130] Wenqi Shi, Sheng Zhou, Zhisheng Niu, Miao Jiang, and Lu Geng. Joint device scheduling and resource allocation for latency constrained wireless federated learning. *IEEE Transactions on Wireless Communications*, 20(1):453–467, 2020.

[131] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[132] Jiangang Shu, Xiaohua Jia, Kan Yang, and Hua Wang. Privacy-preserving task recommendation services for crowdsourcing. *IEEE Transactions on Services Computing*, 14(1):235–247, 2018.

[133] Shivani Singh, Razia Sulthana, Tanvi Shewale, Vinay Chamola, Abderrahim Benslimane, and Biplab Sikdar. Machine-learning-assisted security and privacy provisioning for edge computing: A survey. *IEEE Internet of Things Journal*, 9(1):236–260, 2022.

[134] Mahdi Soleymani, Ramy E Ali, Hessam Mahdavifar, and A Salman Avestimehr. Approxifer: A model-agnostic approach to resilient and robust prediction serving systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8342–8350, 2022.

[135] Kyungrak Son and Wan Choi. Distributed matrix multiplication based on frame quantization for straggler mitigation. *IEEE Transactions on Signal Processing*, 2022.

[136] Yao Song, Liang Wang, Limin Xiao, Wei Wei, Rafał Scherer, Guangjun Qin, and Jinquan Wang. Hypergraph-partitioning-based online joint scheduling of tasks and data. *The Journal of Supercomputing*, pages 1–30, 2022.

[137] Dimitris Stripelis, Paul M Thompson, and José Luis Ambite. Semi-synchronous federated learning for energy-efficient training and accelerated convergence in cross-silo settings. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[138] Abeda Sultana, Md Mainul Haque, Li Chen, Fei Xu, and Xu Yuan. Eiffel: Efficient and fair scheduling in adaptive federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[139] Lili Sun and Hua Wang. Access control and authorization for protecting disseminative information in e-learning workflow. *Concurrency and Computation: Practice and Experience*, 23(16):2034–2042, 2011.

[140] Lili Sun, Hua Wang, Jeffrey Soar, and Chunming Rong. Purpose based access control for privacy protection in e-healthcare services. *Journal of Software*, 7(11):2443–2449, 2012.

[141] Wen Sun, Shiyu Lei, Lu Wang, Zhiqiang Liu, and Yan Zhang. Adaptive federated learning and digital twin for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(8):5605–5614, 2020.

[142] Xiaoxun Sun, Min Li, and Hua Wang. A family of enhanced (l, α)-diversity models for privacy preserving data publishing. *Future Generation Computer Systems*, 27(3):348–356, 2011.

[143] Xiaoxun Sun, Lili Sun, and Hua Wang. Extended k-anonymity models against sensitive attribute disclosure. *Computer Communications*, 34(4):526–535, 2011.

[144] Xiaoxun Sun, Hua Wang, and Jiuyong Li. Satisfying privacy requirements: One step before anonymization. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 181–188. Springer Berlin Heidelberg, 2010.

[145] Xiaoxun Sun, Hua Wang, Jiuyong Li, and Traian Marius Truta. Enhanced p-sensitive k-anonymity models for privacy preserving data publishing. *Transactions on Data Privacy*, 1(2):53–66, 2008.

[146] Xiaoxun Sun, Hua Wang, Jiuyong Li, and Yanchun Zhang. Injecting purpose and trust into data anonymisation. *Computers & security*, 30(5):332–345, 2011.

[147] Xiaoxun Sun, Hua Wang, Jiuyong Li, and Yanchun Zhang. An approximate microaggregation approach for microdata protection. *Expert Systems with Applications*, 39(2):2211–2219, 2012.

[148] Xiaoxun Sun, Hua Wang, Jiuyong Li, and Yanchun Zhang. Satisfying privacy requirements before data anonymization. *The Computer Journal*, 55:422–437, 04 2012.

[149] Afaf Tak and Soumaya Cherkaoui. Federated edge learning: Design issues and challenges. *IEEE Network*, 35(2):252–258, 2020.

[150] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376. PMLR, 2017.

[151] Zhongyun Tang, Haiyang Hu, and Chonghuan Xu. A federated learning method for network intrusion detection. *Concurrency and Computation: Practice and Experience*, 34(10):e6812, 2022.

[152] Yun Teng, Zhiyue Li, Jing Huang, and Guangyan Zhang. Short tail: taming tail latency for erasure-code-based in-memory systems. *Frontiers of Information Technology & Electronic Engineering*, pages 1–12, 2022.

[153] Pu Tian, Weixian Liao, Wei Yu, and Erik Blasch. Wscc: A weight similarity based client clustering approach for non-iid federated learning. *IEEE Internet of Things Journal*, 2022.

[154] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[155] Ryan J Tibshirani. The lasso problem and uniqueness. *Electronic Journal of statistics*, 7:1456–1490, 2013.

[156] Shreshth Tuli, Sukhpal Singh Gill, Peter Garraghan, Rajkumar Buyya, Giuliano Casale, and Nick Jennings. Start: Straggler prediction and mitigation for cloud computing environments using encoder lstm networks. *IEEE Transactions on Services Computing*, 2021.

[157] Raihan ur Rasool, Maleeha Najam, Hafiz Farooq Ahmad, Hua Wang, and Zahid Anwar. A novel json based regular expression language for pattern matching in the internet of things. *Journal of Ambient Intelligence and Humanized Computing*, 10(4):1463–1481, 2019.

[158] Yeshwanth Venkatesha, Youngeun Kim, Hyoungseob Park, Yuhang Li, and Priyadarshini Panda. Addressing client drift in federated continual learning with adaptive optimization. *arXiv preprint arXiv:2203.13321*, 2022.

[159] Madhusanka Manimel Wadu, Sumudu Samarakoon, and Mehdi Bennis. Joint client scheduling and resource allocation under channel uncertainty in federated learning. *IEEE Transactions on Communications*, 69(9):5962–5974, 2021.

[160] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1698–1707. IEEE, 2020.

[161] Hua Wang, Jinli Cao, and Yanchuan Zhang. Ticket-based service access scheme for mobile users. *Aust. Comput. Sci. Commun.*, 24(1):285–292, jan 2002.

[162] Hua Wang, Jinli Cao, and Yanchun Zhang. A flexible payment scheme and its role-based access control. *IEEE Transactions on knowledge and Data Engineering*, 17(3):425–436, 2005.

[163] Hua Wang, Jinli Cao, and Yanchun Zhang. *Access Control Management in Cloud Environments*. Springer, Germany, 2020.

[164] Hua Wang, Yongzhi Wang, Tarek Taleb, and Xiaohong Jiang. Editorial: Special issue on security and privacy in network computing. *World Wide Web*, 23, 07 2019.

[165] Hua Wang, Yanchun Zhang, and Jinli Cao. Effective collaboration with information sharing in virtual universities. *IEEE Transactions on Knowledge and Data Engineering*, 21(6):840–853, 2008.

[166] Hua Wang, Yanchun Zhang, Jinli Cao, and Vijay Varadharajan. Achieving secure and flexible m-services through tickets. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(6):697–708, 2003.

[167] Jingrong Wang and Ben Liang. Gradient and projection free distributed online min-max resource optimization. In *Learning for Dynamics and Control Conference*, pages 391–403. PMLR, 2022.

[168] Qiyuan Wang, Qianqian Yang, Shibo He, Zhiguo Shui, and Jiming Chen. Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation. *arXiv preprint arXiv:2205.13797*, 2022.

[169] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[170] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

[171] Xite Wang, Chaojin Wang, Mei Bai, Qian Ma, and Guanyu Li. Htd: heterogeneous throughput-driven task scheduling algorithm in mapreduce. *Distributed and Parallel Databases*, 40(1):135–163, 2022.

[172] Zhiyuan Wang, Hongli Xu, Jianchun Liu, Yang Xu, He Huang, and Yangming Zhao. Accelerating federated learning with cluster construction and hierarchical aggregation. *IEEE Transactions on Mobile Computing*, 2022.

[173] Zi-Jia Wang, Zhi-Hui Zhan, Ying Lin, Wei-Jie Yu, Hua Wang, Sam Kwong, and Jun Zhang. Automatic niching differential evolution with contour prediction approach for multimodal optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(1):114–128, 2019.

[174] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[175] B. Wolford. What is gdpr, the eu's new data protection law?.

[176] Joel Wolfrath, Nikhil Sreekumar, Dhruv Kumar, Yuanli Wang, and Abhishek Chandra. Haccs: Heterogeneity-aware clustered client selection for accelerated federated learning. In *IEEE IPDPS*, 2022.

[177] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. Fedadapt: Adaptive offloading for iot devices in federated learning. *IEEE Internet of Things Journal*, 2022.

[178] Hongda Wu and Ping Wang. Node selection toward faster convergence for federated learning on non-iid data. *IEEE Transactions on Network Science and Engineering*, 2022.

[179] Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1539–1551, 2020.

[180] Marvin Xhemrishi, Alexandre Graell i Amat, Eirik Rosnes, and Antonia Wachter-Zeh. Computational code-based privacy in coded federated learning. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 2034–2039. IEEE, 2022.

[181] Lin Xiao. Dual averaging method for regularized stochastic learning and online optimization. *Advances in Neural Information Processing Systems*, 22, 2009.

[182] Binhui Xie, Longhui Yuan, Shuang Li, Chi Harold Liu, Xinjing Cheng, and Guoren Wang. Active learning for domain adaptation: An energy-based approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8708–8716, 2022.

[183] Qianqian Xie, Jimin Huang, Min Peng, Yihan Zhang, Kaifei Peng, and Hua Wang. Discriminative regularized deep generative models for semi-supervised learning. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 658–667. IEEE, 2019.

[184] Eunju Yang, Dong-Ki Kang, and Chan-Hyun Youn. Boa: batch orchestration algorithm for straggler mitigation of distributed dl training in heterogeneous gpu cluster. *The Journal of Supercomputing*, 76(1):47–67, 2020.

[185] Haibo Yang, Xin Zhang, Prashant Khanduri, and Jia Liu. Anarchic federated learning. In *International Conference on Machine Learning*, pages 25331–25363. PMLR, 2022.

[186] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[187] Wei Yang, Wei Xiang, Yuan Yang, and Peng Cheng. Optimizing federated learning with deep reinforcement learning for digital twin empowered industrial iot. *IEEE Transactions on Industrial Informatics*, 2022.

[188] Zhengjie Yang, Sen Fu, Wei Bao, Dong Yuan, and Albert Y Zomaya. Fastslowmo: Federated learning with combined worker and aggregator momenta. *IEEE Transactions on Artificial Intelligence*, 2022.

[189] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1998–2007. PMLR, 2018.

[190] Jiao Yin, Ming Jian Tang, Jinli Cao, Hua Wang, and Mingshan You. A real-time dynamic concept adaptive learning algorithm for exploitability prediction. *Neurocomputing*, 472:252–265, 2022.

[191] Jiao Yin, MingJian Tang, Jinli Cao, and Hua Wang. Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description. *Knowledge-Based Systems*, 210:106529, 2020.

[192] Jiao Yin, MingJian Tang, Jinli Cao, Hua Wang, Mingshan You, and Yongzheng Lin. Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning. *World Wide Web*, pages 1–23, 2021.

[193] Jiao Yin, Mingjian Tang, Jinli Cao, Mingshan You, Hua Wang, and Mamoun Alazab. Knowledge-driven cybersecurity intelligence: Software vulnerability co-exploitation behaviour discovery. *IEEE Transactions on Industrial Informatics*, pages 1–9, 01 2022.

[194] Mingshan You, Jiao Yin, Hua Wang, Jinli Cao, and Yuan Miao. A minority class boosted framework for adaptive access control decision-making. In *Web Information Systems Engineering – WISE 2021*, pages 143–157, 01 2021.

[195] Mingshan You, Jiao Yin, Hua Wang, Jinli Cao, Kate Wang, Yuan Miao, and Elisa Bertino. A knowledge graph empowered online learning framework for access control decision-making. *World Wide Web*, pages 1–22, 06 2022.

[196] Liangkun Yu, Rana Albelaihi, Xiang Sun, Nirwan Ansari, and Michael Devetsikiotis. Jointly optimizing client selection and resource management in wireless federated learning for internet of things. *IEEE Internet of Things Journal*, 9(6):4385–4395, 2021.

[197] Syed Zawad, Feng Yan, and Ali Anwar. Systems bias in federated learning. In *Federated Learning*, pages 259–278. Springer, 2022.

[198] Dun Zeng, Siqi Liang, Xiangjing Hu, and Zenglin Xu. Fedlab: A flexible federated learning framework. *arXiv preprint arXiv:2107.11621*, 2021.

[199] Fuyong Zhang, Yi Wang, Shigang Liu, and Hua Wang. Decision-based evasion attacks on tree ensemble classifiers. *World Wide Web*, 23(5):2957–2977, 2020.

[200] Ji Zhang, Xiaohui Tao, and Hua Wang. Outlier detection from large distributed databases. *World Wide Web*, 17(4):539–568, 2014.

[201] Ji Zhang, Hua Wang, Xiaohui Tao, and Lili Sun. Sodit: An innovative system for outlier detection using multiple localized thresholding and interactive feedback. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1364–1367. IEEE, 2013.

[202] Jinghui Zhang, Xinyu Cheng, Cheng Wang, Yuchen Wang, Zhan Shi, Jiahui Jin, Aibo Song, Wei Zhao, Liangsheng Wen, and Tingting Zhang. Fedada: Fast-convergent adaptive federated learning in heterogeneous mobile edge computing environment. *World Wide Web*, pages 1–28, 2022.

[203] Mi Zhang, Qiuping Wang, Zhirong Shen, and Patrick PC Lee. Pocache: Toward robust and configurable straggler tolerance with parity-only caching. *Journal of Parallel and Distributed Computing*, 2022.

[204] Yikai Zhang, Hui Qu, Dimitris Metaxas, and Chao Chen. Local regularizer improves generalization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6861–6868, 2020.

[205] Yu Zhang, Morning Duan, Duo Liu, Li Li, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. Csafl: A clustered semi-asynchronous federated learning framework. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2021.

[206] Yu-Hui Zhang, Yue-Jiao Gong, Ying Gao, Hua Wang, and Jun Zhang. Parameter-free voronoi neighborhood for evolutionary multimodal optimization. *IEEE Transactions on Evolutionary Computation*, 24(2):335–349, 2020.

[207] Yuanyu Zhang, Yulong Shen, Hua Wang, Jianming Yong, and Xiaohong Jiang. On secure wireless communications for iot under eavesdropper collusion. *IEEE Transactions on Automation Science and Engineering*, 13(3):1281–1293, 2016.

[208] Yuanyu Zhang, Yulong Shen, Hua Wang, Yanchun Zhang, and Xiaohong Jiang. On secure wireless communications for service oriented computing. *IEEE Transactions on Services Computing*, 11(2):318–328, 2015.

[209] Zizhen Zhang, Luyao Teng, Mengchu Zhou, Jiahai Wang, and Hua Wang. Enhanced branch-and-bound framework for a class of sequencing problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(5):2726–2736, 2021.

[210] Zhengyi Zhong, Weidong Bao, Ji Wang, Xiaomin Zhu, and Xiongtao Zhang. Flee: A hierarchical federated learning framework for distributed deep neural network over cloud, edge and end device. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[211] Chendi Zhou, Ji Liu, Juncheng Jia, Jingbo Zhou, Yang Zhou, Huaiyu Dai, and Dejing Dou. Efficient device scheduling with multi-job federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9971–9979, 2022.

[212] Qihua Zhou, Song Guo, Haodong Lu, Li Li, Minyi Guo, Yanfei Sun, and Kun Wang. Falcon: Addressing stragglers in heterogeneous parameter server via multiple parallelism. *IEEE Transactions on Computers*, 70(1):139–155, 2020.

[213] Shenglong Zhou and Geoffrey Ye Li. Federated learning via inexact admm. *arXiv preprint arXiv:2204.10607*, 2022.

[214] Xiaokang Zhou, Wei Liang, Jianhua Ma, Zheng Yan, I Kevin, and Kai Wang. 2d federated learning for personalized human activity recognition in cyber-physical-social systems. *IEEE Transactions on Network Science and Engineering*, 2022.

[215] Zihao Zhou, Yanan Li, Xuebin Ren, and Shusen Yang. Towards efficient and stable k-asynchronous federated learning with unbounded stale gradients on non-iid data. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2022.

[216] Chaoyang Zhu, Xiao Zhu, Junyu Ren, and Tuanfa Qin. Blockchain-enabled federated learning for uav edge computing network: Issues and solutions. *IEEE Access*, 2022.

[217] Hongbin Zhu, Miao Yang, Junqian Kuang, Hua Qian, and Yong Zhou. Client selection for asynchronous federated learning with fairness consideration. In *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 800–805. IEEE, 2022.

[218] Jianyong Zhu, Chunming Hu, Tianyu Wo, and Xiaoqiang Yu. Scalereactor: A graceful performance isolation agent with interference detection and investigation for container-based scale-out workloads. *Concurrency and Computation: Practice and Experience*, 34(4):e6666, 2022.

[219] Kun Zhu, Jiawei Liang, Juan Li, and Changyan Yi. Coded distributed computing with predictive heterogeneous user demands: A learning auction approach. *IEEE Journal on Selected Areas in Communications*, 2022.

[220] Ligeng Zhu, Hongzhou Lin, Yao Lu, Yujun Lin, and Song Han. Delayed gradient averaging: Tolerate the communication latency for federated learning. *Advances in Neural Information Processing Systems*, 34:29995–30007, 2021.

[221] Zheqi Zhu, Shuo Wan, Pingyi Fan, and Khaled B. Letaief. Federated multiagent actor–critic learning for age sensitive mobile-edge computing. *IEEE Internet of Things Journal*, 9(2):1053–1067, 2022.

[222] Huiping Zhuang, Zhenyu Weng, Fulin Luo, Toh Kar-Ann, Haizhou Li, and Zhiping Lin. Accumulated decoupled learning with gradient staleness mitigation for convolutional neural networks. In *International Conference on Machine Learning*, pages 12935–12944. PMLR, 2021.

[223] Weiming Zhuang, Xin Gan, Yonggang Wen, and Shuai Zhang. Easyfl: A low-code federated learning platform for dummies. *IEEE Internet of Things Journal*, 2022.

[224] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. Pysyft: A library for easy federated learning. In *Federated Learning Systems*, pages 111–139. Springer, 2021.

[225] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.