EFFICIENT AND ROBUST FEDERATED LEARNING WITH DIVERSE AND DYNAMIC DATA

TIFFANY TUOR

A Thesis Submitted in Fulfilment of Requirements for the Degree of

Doctor of Philosophy of Imperial College London and

Diploma of Imperial College

Communications and Signal Processing Group

Department of Electrical and Electronic Engineering

Imperial College London

2020

Declaration of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Tiffany Tuor

Abstract

Traditional machine learning algorithms require data to be centralized. In practice, data is often generated at multiple locations. Transmitting all the data to a central location is often impractical due to resource constraints but also privacy concerns. Federated learning addresses this problem by allowing clients to collaboratively learn a global model, while keeping their raw data stored locally. Existing federated learning approaches face some limitations when applied to real-world systems. This thesis aims to address some of these limitations, specifically to improve resource efficiency and robustness of federated learning with heterogeneous and dynamic data.

In the first part of this thesis, we optimize resource efficiency and robustness of federated learning under spatial data diversity. We propose an algorithm able to learn systems characteristics in real-time and dynamically adapt the frequency of aggregation to maximize the learning accuracy for a given resource budget. Furthermore, most existing federated learning approaches focus on training a model using predefined datasets at the client nodes. However, the spatial diversity of data collected at each client can often affect model accuracy. To consider this data diversity but also noisy data, we propose an approach to select only relevant data for a given machine learning task.

In the second part of this thesis, we address temporal data heterogeneity, which is caused by clients dropping out of the systems before completion of the learning task due to energy or connectivity constraints at the clients. To this end, we propose a continual learning approach that allows the global model to continuously learn on data changing over time, without forgetting previously learned information. This new continual learning approach is readily applicable to federated learning.

Finally, aiming to address problems related to system heterogeneity, we propose an approach to efficiently monitor and forecast resource utilisation in large-scale and heterogeneous distributed systems.

To my family
for their unconditional love and support

Acknowledgments

Undertaking this PhD has been a life changing experience and it would not have been possible without the support and guidance that I received from many colleagues, friends and family whom I would like to thanks.

First of all, I would like to express my sincere gratitude to my PhD supervisor Professor Kin K. Leung, for his constant guidance and support over these last four years. The lessons I have learned from him are invaluable, as are the skills acquired over these last four years. Kin deeply cares about his student, and always put his students'interest first. He always treated me with the highest degree of respect, honesty, and openness. I will always consider myself very privileged to have been one of his students.

I would also like to express my special thanks to Dr. Shiqiang Wang, my mentor and main collaborator, whose indispensable guidance and patience were continuous throughout my PhD years. From him, I learned not only how to address specific issues in my area of research, but also the way to approach, frame, break down, and present results of complex research problems. I benefited tremendously from Shiqiang's mentorship, and I wish every PhD student to have a mentor like him. I would also like to express my sincere thanks to him for hosting two unforgettable research internships for me at IBM T.J. Watson Research Center in New York during 2017 and 2019, but also for a virtual internship in 2020. I will always keep very good memories from my summers spent in New York. I will particularly remember some very nice moments spent over dinners shared with Shiqiang, Ziyao Zhang, Yuang Jiang, and Hanlin Liu, who introduce me to some delicious Chinese dishes. They made my time there very special and I would like to thank them for making me feel

not only like their colleague but also like their friend.

I would like to acknowledge the funding source of my PhD, namely the International Technology Alliance in Distributed Analytics and Information Sciences (DAIS-ITA) research project which has provided me with precious opportunities to collaborate with many world-class researchers. I take this opportunity to thanks some of DAIS-ITA collaborators, including Dr. Bong Jun Ko, Dr. ChangChang Liu, Dr. Theodoros Salonidis and Dr. Christian Makaya from IBM T.J. Watson Research Center, Dr. Ting He from Penn State University, Dr. Kevin Chan and Dr. Ananthram Swami from the US Army Research Laboratory, and James Lambert and John Melrose from the UK Defence Science and Technology Laboratory. It has been a pleasure, and a great honor to work with you, to receive your invaluable feedbacks, and to publish research papers together.

I would also like to sincerely thanks my two viva examiners Professor Elisa Bertino from Purdue University and Dr. Krystian Mikolajczyk from Imperial College London, for the time they spent on reading my thesis, attending my viva and providing very insightful comments.

Finally, I would like to thanks my family for their constant love and support, and for encouraging me in whatever adventures and challenges I decide to undertake. Each of them is responsible for the completion of my PhD, my Dad, for reminding me every day to keep the big picture in mind and not letting every day life's details drag me down, my Mum, for being the very first one to spark my interest for Computer Science and inspiring me to follow this path, and my sister, the *real* Doctor (of Medicine) of the family, whose determination is inspirational.

I could not end without thanking Soteris, who has been by my side every day during these last years, in both good and challenging times.

Tiffany Tuor

Imperial College London, December 2020

Contents

External Publications

The work reported in this thesis has been published/submitted as the following papers.

1. Federated Learning under Resources Constraints (Chapter 2):
   (a) **T. Tuor**, S. Wang, T. Salonidis, B. J. Ko, K. K. Leung, "Demo abstract: distributed machine learning at resource-limited edge nodes", *in IEEE INFOCOM, Apr. 2018*
   (b) S. Wang, **T. Tuor**, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, "When edge meets learning: adaptive control for resource-constrained distributed machine learnin", *in IEEE INFOCOM, Apr. 2018* (**Travel Grant Award**)
   (c) **T. Tuor**, S. Wang, K. K. Leung, K. Chan, "Distributed machine learning in coalition environments: overview of techniques", *in the 21st International Conference on Information Fusion (FUSION), July 2018*
   (d) S. Wang, **T. Tuor**, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, "Adaptive Federated Learning in Resource Constrained Edge Computing Systems", *IEEE Journal on Selected Areas in Communications (J-SAC), Mar. 2019* (**IEEE Communications Society Leonard G. Abraham Prize, Best paper published in J-SAC in the last 3 years**)
2. Efficient Federated Learning with Diverse Tasks and Data (Chapter 3):
   (a) **T. Tuor**, S. Wang, B. J. Ko, C. Liu, K. K. Leung, "Overcoming Noisy and Irrelevant Data in Federated Learning", *in the 25th International Conference on Pattern Recognition (ICPR), Jan 2021*

3. Forecasting Resources Utilization in Distributed Systems (Chapter 5)
   (a) **T. Tuor**, S. Wang, K. K. Leung, B. J. Ko, "Online collection and forecasting of resource utilization in large-scale distributed systems,", *IEEE International Conference on Distributed Computing Systems (ICDCS), Jul. 2019* (**Travel Grant Award**)

The following publications were completed as part of my Ph.D. study, but its contents are not included in this thesis for compactness.

1. **T. Tuor**, S. Wang, K. K. Leung, B. J. Ko, "Understanding information leakage of distributed inference with deep neural networks: Overview of information theoretic approach and initial results", *in Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, Apr. 2018*
2. D. Conway-Jones, **T. Tuor**, S. Wang, K. K. Leung, "Demonstration of federated learning in a resource-constrained networked environment", *in IEEE International Conference on Smart Computing (SMARTCOMP), Jun. 2019*

List of Figures

Abbreviations

ARIMA	Autoregressive Integrated Moving Average
BNN	Bayesian Neural Network
CL	Continual Learning
CNN	Convolutional Neural Networks
DGD	Deterministic Gradient Descent
FedAvg	Federated Averaging Algorithm
IoT	Internet-of-Things
LSTM	Long Shot Term Memory
RMSE	Root Mean Square Error
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine

CHAPTER 1

Introduction

1.1 Overview

1.1.1 Motivation

To analyze large amounts of data and obtain useful information for the detection, classification, and prediction of future events, machine learning techniques are often applied. One key enabler of machine learning is the ability to learn (train) models using a very large amount of data. Traditional machine learning algorithms require this data to be centralized either on a single machine or within a datacenter. However, in practice, data is often generated at multiple locations (e.g., mobile devices, Internet-of-Things (IoT) gateways or sensors). Transmitting all the data to a central location would consume excessive communication bandwidth, storage and may also violate user privacy regulations.

Federated learning addresses this problem by allowing clients[1] (e.g., mobile devices, IoT gateways or sensors, organization) to collaboratively learn a global model while keeping their raw data stored locally [2]. By enabling analytic over multiple data sources, federated learning opens the door to many promising applications: from analytics in IoT and mobile devices, where data from billions of connected devices is fused to build predictive systems, to collaboration between different organizations such as medical institutions that would like to combine their patient data to

[1] The terms clients, nodes, devices are used interchangeably in this thesis.

improved diagnostic tools, cooperation among pharmaceutical companies for drug discoveries, or even financial institutions that would like to collaborate to combat frauds and money laundering.

An area related to federated learning is distributed machine learning in datacenters through the use of worker machines and parameter servers [24]. The main difference between the datacenter environment and the federated learning environment is that shared storage is usually used in datacenters. The worker machines do not keep persistent data storage on their own, and they fetch the data from the shared storage at the beginning of the learning process. As a result, the workers' data samples are usually independent and identically distributed (i.i.d.). In the section 1.1.3, we will describe some of the challenges that make federated learning settings distinct from other distribution optimization problem such as distributed learning in data center settings.

1.1.2 Federated Averaging Algorithm (FedAvg)

In 2016, authors in [2] proposed the first federated learning algorithm, referred to as the Federated Averaging Algorithm (FedAvg), which aims to fit a single global machine learning model on data store at multiple remote clients under the constraint that data has to remain stored and processed locally. Only intermediate model parameters can be sent to the central server. More specifically, each client performs model training computations locally on its own dataset (i.e.,takes steps of gradient descent on the current model using local data) and sends its local model parameter to the central server. The central server, which is used to orchestrate the learning[2], aggregates the model parameters received from the different clients and send the updated parameter back to the clients for the next round of iterations. The full train-

[2]The aggregator is a logical component that can run on the remote cloud, a network equipment, or one of the edge nodes. Aggregator, server, central node are terms used interchangeably in this thesis

ing process involves several rounds of exchanges between the clients and the central server until the model is fully trained (See additional details in section 2.3.3).

1.1.3 Challenges

FedAvg applied in real-world systems faces some challenges. First, it requires expensive communication as the clients and the server need to frequently communicate to exchange model parameters with often high dimensionality (i.e., state-of-the-art deep neural network (DNN) model have millions of parameters), which makes the process bandwidth consuming. Failing to communicate frequently can significantly deteriorate the training as it might impact the convergence of the global models.

A second challenge comes from the data heterogeneity generated at different clients (spatial data heterogeneity). In practice, the distribution of data across clients is highly non-iid. For example, two clients may write the same letter, but with different kinds of handwriting styles. Clients may also have different label distributions. Some labels might be more present in some clients than others. Some clients may share the same labels, but they correspond to different features at different clients, or the opposite, they share the same features but are labeled in different ways in each client. This spatial data heterogeneity may lead to some scenarios where local model trained simply on local datasets outperforms the global model, which removes the incentive to participate in the federated training.

Finally, system heterogeneity is another significant challenge. As participating devices often have very different hardware (i.e., CPU, memory) and network connectivity (i.e., 3G, 4G, 5G, Wi-Fi), the central server often receives updates from clients at different times asynchronously. This device heterogeneity results in only a subset of clients being active at the same time. Furthermore, as federated learning relies on client status (e.g., idle, charging, or connected to an unmetered network), it is very common for edge devices to drop out due to connectivity or energy con-

straints. As devices drop out/join, data on which the global model is trained varies over time. This introduces the notion of temporal data heterogeneity.

1.2 Research Objectives

This thesis aims to address some of these limitations, more specifically to improve resource efficiency and robustness of federated learning under heterogeneous and dynamic data (i.e., temporal data heterogeneity). In the first part of this thesis, we propose some changes to the original federated learning algorithm (i.e., FedAvg) [2] to make it more resource efficient and robust to spatial data diversity. In a second part of this thesis, we address temporal data heterogeneity, which is caused by clients dropping out the systems before completion of the learning task due to energy or connectivity constraints. Finally, intending to address problems related to system heterogeneity, we propose an approach to efficiently monitor and forecast resource utilization in large scale and heterogeneous distributed systems. Note that the privacy aspect of federated learning will remain out of the scope of this thesis.

In the following, we outline the main gaps between existing work and what we would like to achieve in this thesis. Literature reviews on specific aspects are included in each chapter later on.

1.2.1 Adaptive Federated Learning under Resources Constraints

In the FedAvg algorithm, the global aggregation frequency (i.e., number of local steps between two aggregations) is set before training start and remains fixed during training. This can result in a non-optimal usage of resources such as communication bandwidth, computation, power, available energy etc. Hence, we believe that to use available resources efficiently, it is essential to adapt the global aggregation frequency dynamically. To gain intuition on the relevance of this problem, one can

consider the extreme case where all clients have identical data, then aggregation is not necessary (i.e., local model will be identical to the global model). On the other hand, if the data start changing and become very different across clients, one may need to synchronize more frequently. Hence, our goal is to optimize the FedAvg algorithm by proposing an control algorithm able to learn the system and data characteristics in real time and dynamically adapt frequency of aggregation to maximize the learning accuracy for a given set of resource constraints.

1.2.2 Federated Learning with Diverse Tasks and Data

The FedAvg mostly focuses on training a single model by using pre-defined datasets at client devices. While such scenarios are meaningful, it can be far from other important practical situations where each client can have a large variety of data which may or may not be relevant to the given machine learning task. In addition, there can be multiple learning tasks co-existing at the same time. For example, one task can be to train a deep learning model for classifying different animals. Another task aims to train a shallow model for classifying handwritten digits, while the third task is to learn a linear regression model for sensor measurements, etc. To consider this, a challenge is: *How to support federated learning in such a realistic scenario with diverse tasks and data?* In this thesis, we will consider two aspects of this challenge. The first is *how a client selects relevant data in the federated learning process* for a given machine learning task. The second aspect is, with the co-existence of multiple tasks, *how to schedule these multiple federated learning tasks* so that the model training is the most efficient.

1.2.3 Continual Learning for Federated Learning Settings

In federated learning systems, local training requires clients/devices have adequate bandwidth and energy (e.g., battery power). As a result, due to connectivity or en-

ergy constraints, it is not uncommon for some active devices to drop out at some point before the completion of the learning task. Clients leaving the collaboration can have a major impact on the global model. If the data on the leaving clients is very different from other clients, the knowledge from the missing clients will be forgotten over time due to *catastrophic forgetting* phenomenon (i.e., the global model will be updated without taking into account parameters from the missing clients). In other words, as devices drop out/join, data varies over time (i.e., non-stationary), which causes the global model to forget previously acquired knowledge upon learning new information. Continual learning is a field of machine learning that addresses the catastrophic forgetting problem. However, existing techniques are not suitable for federated learning settings as they make assumptions impractical for federated learning (e.g., storage of raw data at a central location). In this thesis, we aim to develop a continual learning technique that can easily be applied to solve the catastrophic forgetting problem in federated learning settings.

1.2.4 Monitoring Large-Scale Federated Learning Systems

In practical federated learning, due to device heterogeneity, the server often receives updates from clients asynchronously. Consequently, a client with few available resources can significantly slow down the whole federated learning process. To mitigate this so-called *straggler effect*, it can be useful to select only a subset of devices to participate in the training. To select the subset of devices (i.e., to minimize training latency), the central server need to assess if the clients will meet certain criteria such as hardware characteristic and connectivity. Hence, the central server need to monitor and predict the resource utilization and availability at each client. However, there exist several challenges for the central node to collect and forecast resource utilization at each client/machine in such large-scale distributed systems. First, it is often bandwidth-consuming and unnecessary to transmit all the measurement data

collected at local nodes to the central node. Second, predictive models for data forecasting typically have high complexity, thus running a forecasting model for the time-series measurement data collected at each local node would consume too much computational resource. Third, measurements at each local node are collected in an online manner, which form a time series; decisions related to data collection and forecasting need to be made in an online manner as well. This thesis will address the above challenges and propose a mechanism that efficiently collects and forecasts the resource utilization at each client/device in a large-scale distributed system. Federated learning systems can further use the results provided by our mechanism for system management.

1.3 Summary of Contributions

The main contributions of this thesis are summarized as follows:

- We propose a control algorithm for federated learning that learns the data distribution, system dynamics, and model characteristics, based on which it dynamically adapts the frequency of global aggregation in real-time to minimize the learning loss under a fixed resource budget. We evaluate the performance of the proposed control algorithm via extensive experiments using real datasets both on a hardware prototype and in a simulated environment, which confirm that our proposed approach to federated learning provides near-optimal performance for different data distributions, various machine learning models, and settings with different numbers of edge nodes (Chapter 2).

- We propose a novel method for identifying the relevant data at each client for a given federated learning task. Our approach first trains a benchmark model using the benchmark data provided by the model requester. By sharing the benchmark model with clients, each client identifies a subset of data

that will be involved in this specific task, where different subsets of data may be involved in different tasks. Our proposed approach works in a distributed manner without requiring clients to share their raw data. We evaluate the performance of our proposed approach by extensive experiments. Using a variety of real-world datasets with different types of noise, we show that our data selection approach outperforms other baseline approaches even when the model requester only provides a very small amount of benchmark data (Chapter 3).

- We formulate the problem of scheduling multiple federated learning tasks with the goal of minimizing the completion time of every training round as a mixed-integer linear program (MILP). We show that this problem is a variant of the NP-hard flow-shop scheduling problem [3] and provide an approximate solution by relaxing and rounding the binary variables. Using a combination of real-world measurements and simulations, we show that our proposed scheduling mechanism is advantageous when compared to other methods as well (Chapter 3).

- We introduce the first task-free continual learning approach that does not require storing training data, making it applicable to federated learning. Our method, based on Bayesian Neural Networks (BNNs), is able to continually learn on new data while minimally forgetting what has been learned previously. Our approach automatically detects shifts in data, which allows the algorithm to work without assuming known task boundaries. Furthermore it ensures that the global model remains within a maximum size so that we do not exceed the storage capacity. We validate our approach on different continual learning scenarios with real world datasets (Chapter 4).

- We present a novel mechanism that allows the central server to efficiently collects and forecasts the resource utilization at each client/device in a large-scale

federated learning system. Extensive experiments of our proposed mechanism have been conducted using three real-world computing cluster datasets, to show the effectiveness of our proposed approach (Chapter 5).

1.4 Organization of the Thesis

This thesis is organized as follows. Chapter 2 introduces the basic of federated machine learning and propose an algorithm for adaptive federated learning under resources constraints. Chapter 3 presents an approach for federated learning with diverse tasks and data. In Chapter 4, an approach for continual learning applicable to federated learning setting, is introduced. Chapter 5 presents an approach to efficiently monitoring and forecasting resource utilization in large-scale distributed systems. Finally, Chapter 6 draws conclusions and discusses future directions of this work. Each chapter contains a summary of the main notation.

CHAPTER 2

Adaptive Federated Learning under Resources Constraints

2.1 Introduction

In a distributed edge environment, training a learning model based on data spread across nodes can be challenging. A federated learning system requires a significant amount of resources to process distributed data and exchange updates to keep the global model sufficiently consistent with all the local datasets. Typically, the federated learning process includes *local update* steps where each edge node/client[1] performs gradient descent to adjust the model parameter to minimize the loss function defined on its own dataset. It also includes *global aggregation* steps where model parameters obtained at different edge nodes are sent to an aggregator which averages these parameters and sends the updated parameter back to the edge nodes for the next iteration. Each local update consumes computation resource at the edge node, and each global aggregation consumes communication resources. In the original federated learning algorithm [2], the global aggregation frequency (i.e., the number of local updates between two aggregations) is set before training start and remains fixed during training. This can result in a non-optimal use of resources, including possible waste of a significant amount of resources. In edge computing environments, where computation and communication resources are scarce, it is necessary to limit

[1] Node, client, devices are used interchangeably

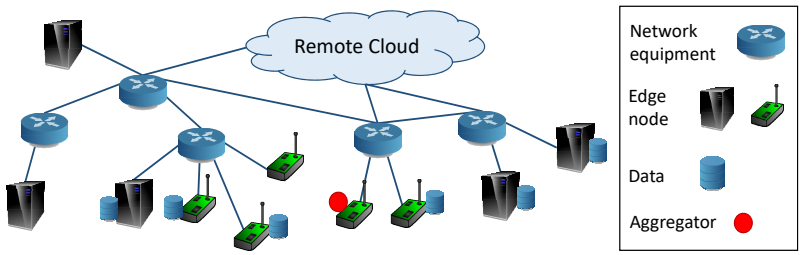


Figure 2.1: System architecture.

the amount of resources used for learning the model.

We proposed here a control algorithm for distributed learning derived from theoretical analysis, which learns the system and data characteristics in real-time and dynamically determines the global aggregation frequency to maximize the learning accuracy for a given resource budget. We focus on the gradient-descent based learning algorithms, which are widely applied in many machine learning tasks, including the training of popular deep neural networks [4]. We consider a typical edge computing environment where edge nodes are interconnected with the remote cloud via network elements (such as gateways and routers), as illustrated in Fig. 2.1.

The rest of this chapter is organized as follows. In the next section, we review the related work. In Section 2.3 we summarize the basics of distributed machine learning. Section 2.4 formulates the problem of federated learning with resource-constrains and proposes a control algorithm to optimize the FedAvg. Experimentation results are shown in Section 2.5 and the conclusion is presented in Section 2.6. A summary of notations used in this chapter is presented in Table 2.2.

2.2 Related Work

Distributed machine learning based on gradient descent has been studied from a theoretical angle in [5, 6, 7], where asymptotic bounds on the training convergence and communication cost are obtained. The result was then extended to general classes

of distributed learning approaches in [8]. To some extent, these results characterize the communication and computation trade-off. However, none of them consider the adaptation of the number of local updates between two global aggregations to achieve a balanced trade-off while considering the learning convergence. Some of the analysis also have unrealistic assumptions such as i.i.d. data distribution at different nodes [5, 6], whereas the more general case involving non-i.i.d. data distributions is much harder to analyze. From the practical perspective, [2] proposes a variant of distributed gradient descent where the global aggregation is performed in a synchronous manner. Experiments using various datasets confirmed the effectiveness of this approach. The number of local updates before each global aggregation is fixed in [2]. It does not provide any theoretical guarantees and the experiments were not conducted in a network setting.

In contrast to the above research, our work addresses the problem of dynamically deciding whether to perform local update or global aggregation based on real-time observations during the distributed learning process. This is a non-trivial problem due to our non-i.i.d. assumption and the complex dependency between each learning step and its previous learning steps, both of which are hard to capture analytically. It is also challenging due to different data distributions at different nodes and the real-time dynamics of the system.

Our main contributions in this chapter are as follows:

1. Based on the theoretical results reported in [9], we propose a control algorithm that learns the data distribution, system dynamics, and model characteristics, based on which it dynamically adapts the frequency of global aggregation in real time to minimize the learning loss under fixed resource budget.
2. We evaluate the performance of the proposed control algorithm via extensive experiments using real world datasets both on a hardware prototype and in a simulated environment, which confirm that our proposed approach pro-

Table 2.1: Loss functions for popular machine learning models

Model	Loss function $f(\mathbf{w}, \mathbf{x}_j, y_j)$ ($\triangleq f_j(\mathbf{w})$)
Smooth SVM	$\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\max\left\{0; 1 - y_j\mathbf{w}^{\mathrm{T}}\mathbf{x}_j\right\}^2$ (λ is const.)
Linear regression	$\frac{1}{2}\|y_j - \mathbf{w}^{\mathrm{T}}\mathbf{x}_j\|^2$
K-means	$\frac{1}{2}\min_l \|\mathbf{x}_j - \mathbf{w}_{(l)}\|^2$ where $\mathbf{w} \triangleq [\mathbf{w}_{(1)}^{\mathrm{T}}, \mathbf{w}_{(2)}^{\mathrm{T}}, ...]^{\mathrm{T}}$
Convolutional neural network	Cross-entropy on cascaded linear and non-linear transforms, see [4]

vides near-optimal performance for different data distributions, various machine learning models, and settings with different numbers of edge nodes.

2.3 Distributed Learning Overview

2.3.1 Background : Loss function

In machine learning, a training data sample j usually consists of two parts. One is a vector $\mathbf{x}_j$ representing the input of the machine learning model, and the other is a scalar y_j representing the target output of the model. To facilitate the learning, each model has a loss function defined on its parameter vector $\mathbf{w}$ for each data sample j. The loss function captures the error of the model on the training data, and the model learning aims to minimise the loss function on a collection of training data samples. For each data sample j, we define the loss function as $f(\mathbf{w}, \mathbf{x}_j, y_j)$, which we abbreviate as $f_j(\mathbf{w})$ in short. Thus, a machine learning process can be seen as an optimization problem where the goal is to minimise the loss function. Examples of loss functions of popular machine learning models are summarised in Table 2.1.

2.3.2 Distributed Loss function

Assume that we have N edge nodes with local datasets $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_i, ..., \mathcal{D}_N$. For each dataset $\mathcal{D}_i$ at node i, the loss function on the collection of data samples at this

node is

$$F_i(\mathbf{w}) \triangleq \frac{1}{|\mathcal{D}_i|} \sum_{j \in \mathcal{D}_i} f_j(\mathbf{w}) \tag{2.1}$$

where $f_j(\mathbf{w})$ is the individual loss for a data sample j as defined above. We define $D_i \triangleq |\mathcal{D}_i|$, where $|\cdot|$ denotes the size of the set, and $D \triangleq \sum_{i=1}^{N} D_i$. Assuming $\mathcal{D}_i \cap \mathcal{D}_{i'} = \emptyset$ for $i \neq i'$, we define the global loss function on all the distributed datasets as

$$F(\mathbf{w}) \triangleq \frac{\sum_{j \in \cup_i \mathcal{D}_i} f_j(\mathbf{w})}{|\cup_i \mathcal{D}_i|} = \frac{\sum_{i=1}^{N} D_i F_i(\mathbf{w})}{D} \tag{2.2}$$

which is equal to the weighted average of the local losses at each edge node i given in (2.1).

The learning problem is to find

$$\mathbf{w}^* = \arg\min F(\mathbf{w}), \tag{2.3}$$

The goal is to solve the learning problem without sending the data to a central place.

2.3.3 Distributed Gradient Descent

A typical way of solving (2.3) is to use the federated averaging algorithm (FedAvg) [2] described as follows.

To start, the synchronization node [2] sends the model parameter $\mathbf{w}(t)$ to all clients (step (1) in Figure 2.2). Each client i sets its local parameter $\mathbf{w}_i(t)$ to be $\mathbf{w}(t)$ and sets the iteration index $t = 0$ at initialization. After that, each node i performs τ steps of gradient descent on the loss function defined on the local dataset $\mathcal{D}_i$, and updates the local model weight $\mathbf{w}_i$ as

$$\mathbf{w}_i(t) = \mathbf{w}_i(t-1) - \eta \nabla F_i(\mathbf{w}_i(t-1)). \tag{2.4}$$

[2] The terms synchronization node, aggregator and server are used interchangeably.

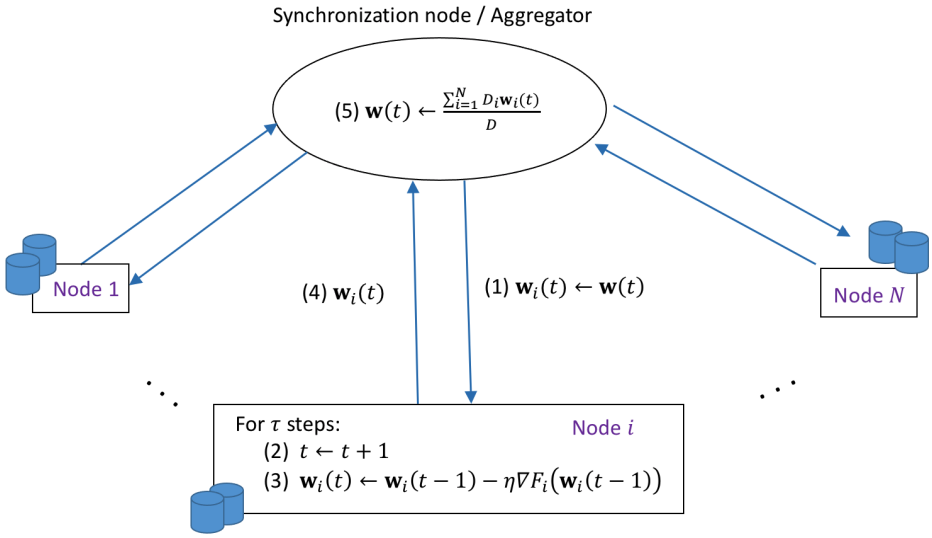


Figure 2.2: Federated Averaging (FedAvg) Algorithm

as shown in steps (2) and (3) of Figure 2.2. The resulting local parameter $\mathbf{w}_i(t)$ from each client is sent to the synchronization node (step (4) Figure 2.2) which compute the weighted average/aggregate of all local parameters:

$$\mathbf{w}(t) = \frac{\sum_{n=1}^{N} |\mathcal{D}_i| \, \mathbf{w}_i(t)}{|\mathcal{D}|} \tag{2.5}$$

where $|\mathcal{D}| = \sum_{n=1}^{N} |\mathcal{D}_i|$ (step (5) in Figure 2.2). Finally, the value of $\mathbf{w}(t)$ is sent back to each node so that the local weights $\{\mathbf{w}_i(t) : \forall n\}$ are updated. The new value of $\mathbf{w}_i(t)$ is then used for the next round of local updates. This process is repeated until a stopping condition is reached. This stopping condition can either be a maximum number of round, a resource budget or when the decrease of the global loss function is smaller than a predefined threshold. If we assume that the training process ends after K synchronizations, then $T = K\tau$ local iterations have been performed in total, and the final model $\mathbf{w}(T)$ is obtained after the last synchronization.

It can be shown that when $\tau = 1$, i.e., when synchronization is performed after every local iteration, distributed gradient descent provides the same mathematical progression as centralized gradient descent [10]. When $\tau > 1$, i.e., when there are

multiple local iterations between synchronization, distributed and centralized gradient descents are in general *not* equivalent. This is because starting from the second step of local iteration after synchronization, the local gradients at different nodes are computed at different model weights (obtained from the first step of local iteration), thus the average of these gradients may not match with any gradient of the global loss function.

2.4 Federated Learning with Resource Constraints

2.4.1 Problem Formulation

A federated learning system requires a significant amount of resources to run over distributed data and exchange updates, to keep the global model sufficiently consistent with all the local datasets. In the FedAvg [2] described in Section 2.3.3, τ is set before training start and remain fixed during training. This can result in a non-optimal usage of resources, which may cause significant waste. In edge computing environments, where computation and communication resources are scarce, it is necessary to limit the amount of resources used for learning the model. Therefore, a natural question in federated learning is how to make efficient use of a given amount of resources for minimizing the loss function of model training. For the FedAvg [2] learning approach presented above, the question narrows down to determining the optimal value of T and τ, so that the global loss function $F(\mathbf{w})$ is minimized without exceeding the resource budget. A possible way to reduce the consumption of resources is to control the number of local iterations in each round (i.e., the value of τ) in order to find the best trade-off between communication and computation resources.

Assuming that each step of local update at *all* participating nodes consumes c units of resource, and each step of global aggregation consumes b units of resource,

where $c \geq 0$ and $b \geq 0$ are both real numbers. For given T and τ, the total amount of consumed resource is then $T\left(c + \frac{b}{\tau}\right)$. Let R denote the total resource budget. It is then possible to formulate a resource-constrained federated learning problem as follows:

$$\begin{aligned} \min_{\tau, T} \quad & F(\mathbf{w}(T)) \\ \text{s.t.} \quad & T\left(c + \frac{b}{\tau}\right) \leq R, \end{aligned} \tag{2.6}$$

Note that we do not distinguish among different types of resource in the formulation. The resource can be time, energy, monetary cost or a combination of them.

To solve (2.6), we need to find out how the values of τ and T affect the loss function (after T iterations), $F(\mathbf{w}(T))$. It is generally impossible to find an exact analytical expression to relate τ and T with $F(\mathbf{w}(T))$, because it depends on the convergence property of gradient descent (for which only upper/lower bounds are known [11]) and the impact of the global aggregation frequency. Further, the resource consumptions c and b can be time-varying in practice, which makes the problem even more challenging than (2.6) alone.

2.4.2 Approximate Solution to (2.6)

Before proposing our approximate solution to (2.6)), we introduce some definitions and assumptions that are required for deriving the following results.

Assumption 1. *The following is assume for all i:*

1. *$F_i(\mathbf{w})$ is convex,*
2. *$F_i(\mathbf{w})$ is ρ-Lipschitz, i.e., $\|F_i(\mathbf{w}) - F_i(\mathbf{w}')\| \leq \rho \|\mathbf{w} - \mathbf{w}'\|$ for any $\mathbf{w}, \mathbf{w}'$,*
3. *$F_i(\mathbf{w})$ is β-smooth, i.e., $\|\nabla F_i(\mathbf{w}) - \nabla F_i(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|$ for any $\mathbf{w}, \mathbf{w}'$.*

The above assumption is satisfied for smooth-SVM and linear regression (see Table 2.1). Even if convexity assumption is required to derive theoretical results, we

will see that in practice this assumption can be relaxed if the loss function is locally convex.

Lemma 1. *$F(\mathbf{w})$ is convex, ρ-Lipschitz, and β-smooth.*

Proof. This is straightforward from Assumption 1, the definition of $F(\mathbf{w})$ in (2.2), and triangle inequality. □

Definition 1. *(Gradient Divergence) For any i and $\mathbf{w}$, we define δ_i as an upper bound of $\|\nabla F_i(\mathbf{w}) - \nabla F(\mathbf{w})\|$, i.e.,*

$$\|\nabla F_i(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta_i \tag{2.7}$$

We also define $\delta \triangleq \frac{\sum_i D_i \delta_i}{D}$.

This metric captures the divergence between the gradient of a local loss function and the gradient of the global loss function. This divergence is related to how the data is distributed at different nodes.

Based on this assumption and definition, it has been shown in [9], that an approximation to solve (2.6) can be found by solving:

$$\tau^* = \arg\max_{\tau} G(\tau) \triangleq \arg\max_{\tau} \frac{\tau}{\tau + a} \left(\eta \left(1 - \frac{\beta\eta}{2} \right) - \frac{\varphi h(\tau)}{\tau} \right) \tag{2.8}$$

where

$$h(\tau) \triangleq \frac{\delta}{\beta} \left((\eta\beta + 1)^{\tau} - 1 \right) - \eta\delta\tau \tag{2.9}$$

where $a \triangleq \frac{b}{c}$ is the relative resource consumption of global aggregation normalized by that of local update, η is the gradient-descent step size, which is pre-specified and known, and φ defines the control parameter that is manually chosen and remains fixed for the same machine learning model. Once τ^* is found, we can easily obtain $T^* = \frac{R\tau^*}{c\tau^* + b}$. Furthermore, $G(\tau)$ has a unique maximum [9].

The first-order derivative of $G(\tau)$ is in transcendental form and hence there is no closed-form expression for τ^*. Because τ takes integer values, we can solve for τ^* using a binary search procedure similar to that in [12] with a complexity of $O(\log \tau_{\max})$, where $\tau_{\max}$ is the maximum value of τ in the search, which will be discussed further in the next subsection.

2.4.3 Dynamic Control algorithm

In practice, computation of τ^* has to be integrated into the FedAvg algorithm. However, the expression of $G(\tau)$ (which includes $h(\tau)$) has unknown parameters. Among these parameters, c and b (and thus a) are related to the system conditions and β and δ are related to the loss function.

The values of c and b are estimated based on measurements of resource consumptions at edge nodes and the aggregator. The estimation method depends on the specific type of resource. For example, when the resource is energy, the sum energy consumption (per local update) at all nodes is considered as c; whereas when the resource is time, the maximum computation time (per local update) at all nodes is considered as c. The parameters β and δ are estimated by combining local estimations :

$$\hat{\beta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\beta}_i}{D} \tag{2.10}$$

where $\hat{\beta}_i \leftarrow \|\nabla F_i(\mathbf{w}_i(t)) - \nabla F_i(\mathbf{w}(t))\| \,/\, \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$, and

$$\hat{\delta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\delta}_i}{D} \tag{2.11}$$

where $\hat{\delta}_i \leftarrow \|\nabla F_i(\mathbf{w}(t_0)) - \nabla F(\mathbf{w}(t_0))\|$. Note that $\hat{\beta}_i$ and $\nabla F_i(\mathbf{w}(t_0))$ are estimated at each node i.

In the following, we describe the exchange protocol which enables the federated learning process to adapt τ to system conditions and different data distributions. The

protocol enables the nodes and the aggregator to coordinate for adjusting the frequency of aggregating local model weights, or equivalently, finding the optimal number of local updates (denoted by τ) between two global aggregations. The value of τ is recomputed during each global aggregation step, based on the most updated parameter estimations. The aggregator monitors the total resource consumption based on these estimates and compares the total resource consumption against the resource budget R. If the consumed resource reaches the budget limit, it stops the learning and returns the final result. The different steps for finding optimal τ are illustrated in Figure 2.3 and described as follows:

- The aggregator starts the learning process by sending an initial message to each node. The initial message contains the initial weight $\mathbf{w}(0)$ and additional information such as the step size and the batch size (Step 1 & Step 1').
- Each node i set its local parameters to $\mathbf{w}(0)$ and performs τ initial local updates (Step 2). The local weight obtained after τ iterations is sent back to the aggregator (Step2'). Note that, at this stage, the node is not able to perform parameter estimations (i.e., $\hat{\beta}_i$ and $\nabla F_i(\mathbf{w}(t_0))$) which require to have access to both local model parameters and global model parameter for the same iteration. The global parameter is only available after the first global aggregation. Hence, estimated parameters are only available starting from the following message received from the aggregator.
- After receiving local weights $\mathbf{w}_i(t)$ from each node, the server can aggregate these local weights to obtain the global weight $\mathbf{w}(t)$ (Step 3), which is then sent back to each node, together with a predefined value of τ as the aggregator is still not able to estimate optimal τ (Step 3').
- Each node i set its local parameters $\mathbf{w}_i(t) = \mathbf{w}(t)$ and perform τ local updates. The node estimates the resource consumption for one local update $\hat{c}_i$.

Also, this time each node is able to estimate $\hat{\beta}_i$, $\nabla F_i(\mathbf{w}(t_0)$ where t_0 is the iteration index of the last aggregation (Step 4). Then the node sends back a message with the updated local parameter $\mathbf{w}_i(t)$, the estimated parameters (i.e., $\hat{\beta}_i$, $\nabla F_i(\mathbf{w}(t_0))$ and D_i (i.e., number of data points at node i) (Step 4').

- The aggregator can once again compute the global weight by aggregating the local weights. Furthermore, the agggregator can now estimate $\hat{\beta}$ and $\hat{\delta}$ according to Equations (2.10) and (2.11) and consequently compute the optimal value of τ according to equation (2.8) via binary search on integer values within $[1, \tau_{\text{max}}]$, where we set $\tau_{\text{max}} \leftarrow \gamma\tau$ (Step 5) [3]. The aggregator can then send an updated global $\mathbf{w}(t)$ to all node with estimated τ (Step 5').

- Steps 4, 4', 5 and 5' are repated in loop until $s + \hat{c}\tau + \hat{b} \geq R$

- When $s + \hat{c}\tau + \hat{b} \geq R$, the aggregator sends a last message to the nodes, with a τ that is within the ressource budget (Step 6).

- Each node performs the last round of local updates (Step 7) and sends its local weights to the server for the last time (Step 7').

- The aggregator does the last aggregation and output the final $\mathbf{w}(T)$ (Step 8).

Note that in the exchange protocol described above, we do not explicitly talk about the local and global loss function (Equations (2.1) and (2.2)) because their values are not required to estimate the optimal τ (i.e., only the local and global gradients are needed). However in order to track the value of the global loss function, it is implicitly assumed that each node will send its loss function value (i.e., $F_i(\mathbf{w})$)) to the aggregator after performing the round of updates. The aggregator averages these local loss functions to global loss function value (i.e., $F(\mathbf{w})$).

[3] We search for new values of τ up to γ times the current value of τ, and find the value that maximizes $G(\tau)$, where $\gamma > 0$ is a fixed value. The presence of γ limits the search space and also avoids τ from growing too quickly as initial parameter estimates may be inaccurate

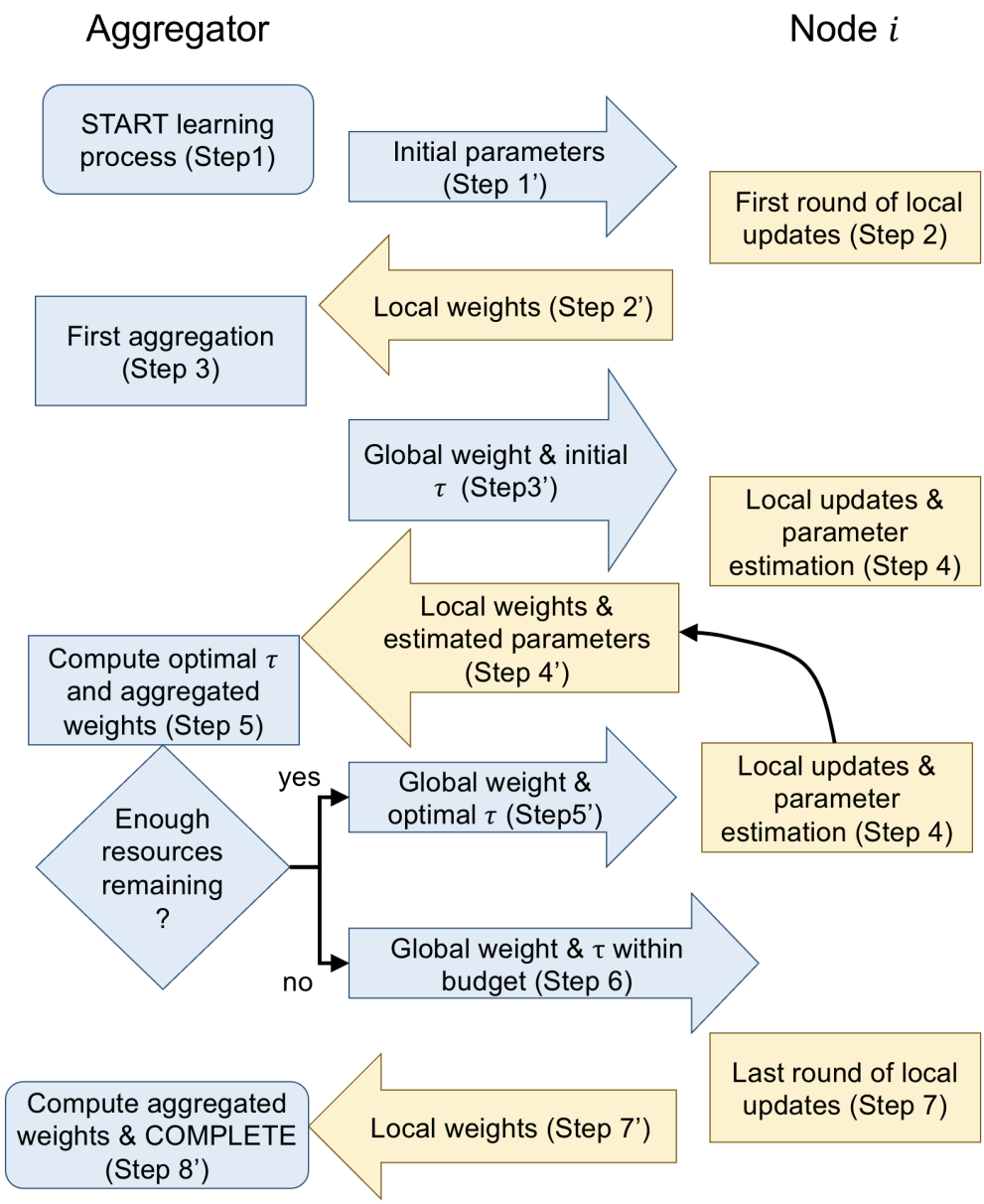


Figure 2.3: Protocol for finding optimal τ

2.5 Experimentation Results

To evaluate the performance of our proposed control algorithm, we conduct a large number of experiments, both on a networked prototype system with 5 nodes and in a simulated environment with the number of nodes varying from 5 to 500.

The prototype system consists of three Raspberry Pi (version 3) devices and two laptop computers, which are all interconnected via Wi-Fi in an office building (see Figure 2.4). This represents an edge computing environment where the computational capabilities of edge nodes are heterogeneous. All these 5 nodes have local datasets (described below) on which model training is conducted. The aggregator is located on one of the laptop computers, and hence co-located with one of the local datasets.

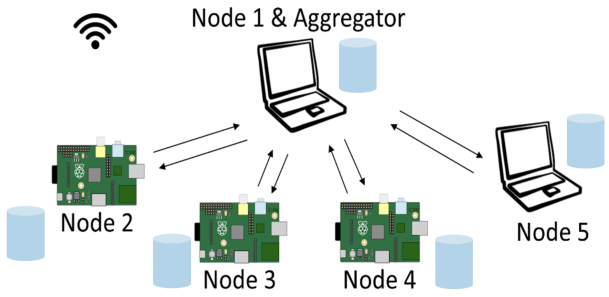


Figure 2.4: Prototype System

We consider time as the resource in our experiments. For the prototype system, we train each model for a fixed amount of time budget. The values of c and b correspond to the actual time used for each local update and global aggregation, respectively. The simulation environment performs model training with simulated resource consumptions, which are set to be equal to the average values of the measurements from the prototype.

We evaluate the training of four different models on three different datasets. The models include smooth support vector machine (SVM), linear regression, K-means, and deep convolutional neural networks (CNN)[4]. See Table 2.1 for a summary of the loss functions of these models, and see [13, 4] for more details. Among these models, the loss functions for smooth-SVM (which we refer to as SVM in short in

[4]The CNN has 7 layers with the following structure: $5 \times 5 \times 32$ Convolutional $\rightarrow 2 \times 2$ MaxPool $\rightarrow 5 \times 5 \times 32$ Convolutional $\rightarrow 2 \times 2$ MaxPool $\rightarrow 1568 \times 256$ Fully connected $\rightarrow 256 \times 10$ Fully connected $\rightarrow$ Softmax.

the following) and linear regression satisfy Assumption 1, whereas the loss functions for K-means and CNN are not convex and thus do not satisfy Assumption 1.

When the amount of training data is large, it is usually computationally prohibitive to compute the gradient of the loss function defined on the entire (local) dataset. In such cases, stochastic gradient descent (SGD) is often used [13], which uses the gradient computed on the loss function defined on a randomly sampled subset of data to approximate the real gradient. We consider both DGD and SGD in the experiments to evaluate the general applicability of the proposed algorithm.

SVM and CNN are trained on the MNIST dataset [14], which contains $70,000$ handwritten digits ($60,000$ for training and $10,000$ for testing). For SVM, we use even and odd digits as a binary label, and we use multi-class labels of 10 different digits for CNN. The SVM training via DGD only uses $1,000$ training and $1,000$ testing data samples out of the entire dataset, because DGD cannot process a large amount of data. The SGD variant of SVM and CNN uses the entire MNIST dataset. Linear regression is performed on Facebook metrics dataset [15], which has 500 samples with multiple attributes related to posts published on a cosmetics brand. The model finds a linear relationship between the total interaction number and all other attributes. K-means is performed on the user knowledge modeling dataset [16], which has 403 samples each with 5 attributes summarizing the user interaction with a web environment. The samples can be grouped into 4 clusters representing different knowledge levels, but we assume that we do not have prior knowledge of such grouping.

We consider four different ways of distributing the data into different nodes. In *Case 1*, each data sample is randomly assigned to a node, thus each node has unbiased (but not full) information. In *Case 2*, all the data samples in each node have the same label[5]. This represents the case where each node has biased information,

[5]When there are more labels than nodes, each node may have data with more than one label, but the number of labels at each node is no more than the total number of labels divided by the total

because the entire dataset has samples with multiple different labels. In *Case 3*, each node has the entire dataset (thus full information). In *Case 4*, data samples with the first half of the labels are distributed as in Case 1; the other samples are distributed as in Case 2. This represents a combined biased and unbiased case.

In all our experiments, we set the search range parameter $\gamma = 10$. Unless otherwise specified, we set the control parameter $\varphi = 0.2$ for SVM, linear regression, and K-means, and we set $\varphi = 10^{-4}$ for CNN. The gradient descent step size is $\eta = 0.01$ for SVM, linear regression, and CNN, and $\eta = 0.1$ for K-means.

In our first set of experiments, the SVM, linear regression, and K-means models were trained on the prototype system, where the resource (time) budget for each model training instance is fixed to 15 seconds. Due to the resource limitation of Raspberry Pi devices, the CNN model was trained in a simulated environment of 5 nodes, with a total budget equivalent to 200 local updates, and a global aggregation consuming $a = 5.0$ times the resource for one local update. The value $a = 5.0$ was obtained from the time measurements of SVM (SGD) with distributed training on MNIST data.

We compare the loss function values of our proposed algorithm (with adaptive τ) to baseline approaches that include centralized training and distributed training with fixed τ. We also compare the classification accuracies for the SVM and CNN classifiers. The centralized baseline is obtained in a simulated environment by running only local updates on a single node subject to the total resource budget, where the local update resource consumption is estimated based on measurements on the prototype system. The distributed baselines run on the prototype system (except for CNN as discussed earlier) under the same setup but with different fixed values of τ (i.e., no adaptation). *This is the same as the state-of-the-art approach in [2].* When fixing $\tau = 1$, it is also equivalent to the synchronous SGD approach in [17].

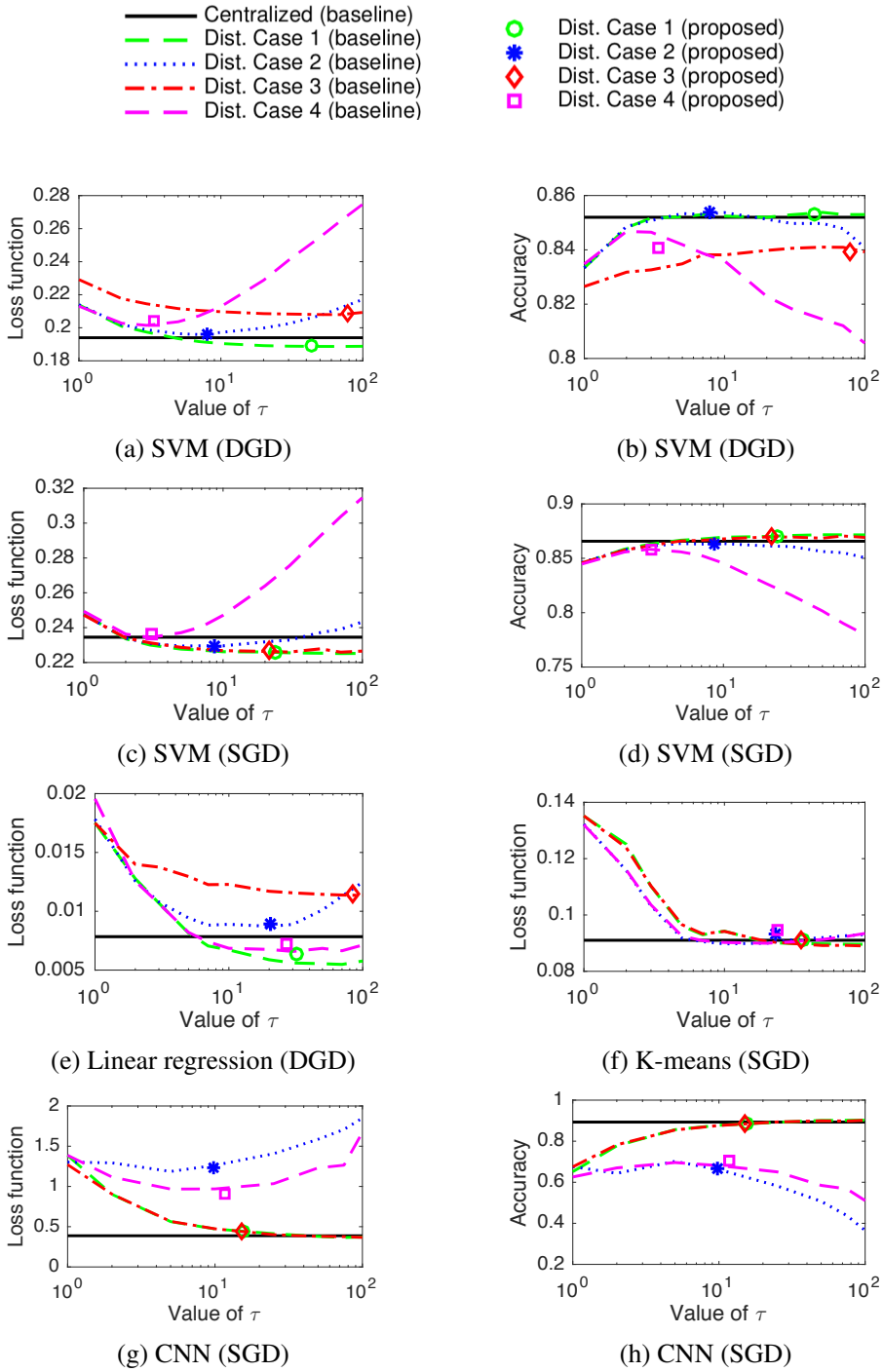
number of nodes rounded to the next integer.

The average results of 30 different experiments (15 for CNN) are shown in Fig. 2.5. We note that *the proposed approach only has one data point (represented by a single marker in the figure) in each case* because the value of τ is adaptive and the average τ is shown in the plot. The centralized case also only has one data point but we show a flat line across different values of τ for the ease of comparison.

We see that the proposed approach performs close to the optimal point in different cases and different models, whereas the optimal value of τ varies from case to case and from model to model so a fixed value of τ does not work well for all cases. Overall, we also note that for the more heterogeneous cases (i.e., Case 2 and Case 4) the optimal value of τ is smaller than for cases where data distribution is more homogeneous (i.e., Case 1 and Case 3). Meaning that the more diverse is the data at different nodes, the more frequently one need to aggregate to insure convergence. We also see that the distributed approach can perform better than the centralized approach for properly chosen values of τ, because for a given amount of time budget, distributed learning is able to make use of the computation resource at multiple nodes, hence improving the training performance. For DGD approaches, Case 3 does not perform as well as Case 1, because the amount of data at each node in Case 3 is larger than that in Case 1, and DGD processes the entire amount of data so Case 3 requires more time for each local update.

Note that even if CNNs do not verify Assumption 1 as the loss function of CNNs isn't convex, experimental results (Fig. 2.5g and 2.5h) show good performance of our approach. This is due to the local convexity of CNNs. Hence Assumption 1 is required in order to theoretically derived Equation (2.8) but can be relaxed in practice if the loss function is locally convex.

Results of SVM (SGD) for the number of nodes varying from 5 to 500 are shown in Fig. 2.6, which are obtained in the simulated environment with $a = 5.0$ and the total budget equal to 868 local updates (both parameters are obtained from mea-



(a) SVM (DGD)

(b) SVM (DGD)

(c) SVM (SGD)

(d) SVM (SGD)

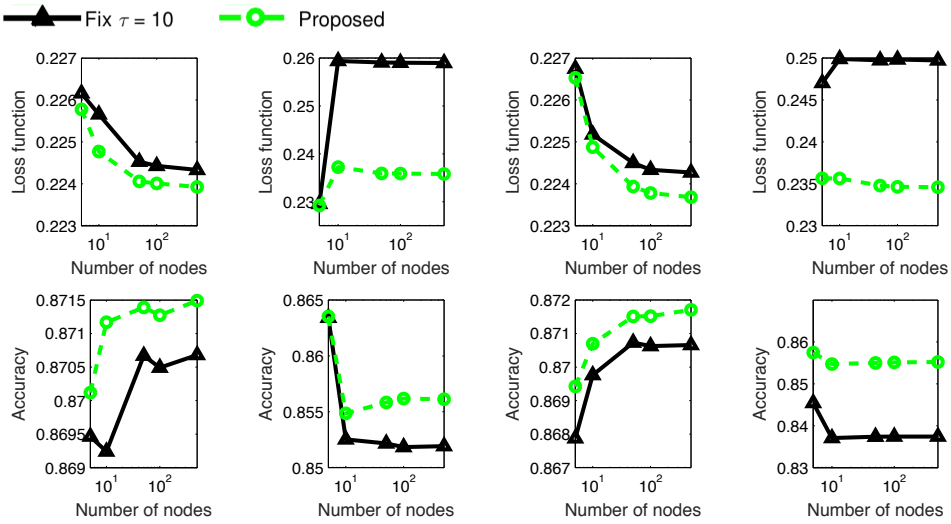
(e) Linear regression (DGD)

(f) K-means (SGD)

(g) CNN (SGD)

(h) CNN (SGD)

Figure 2.5: Loss function values and classification accuracy with different τ. Only SVM and CNN classifiers have accuracy values. The curves show the results from the baseline with different fixed values of τ. Our proposed solution (represented by a single marker for each case) yields an average τ and loss/accuracy that is close to the optimum in all cases.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Figure 2.6: Loss function values and classification accuracy with different numbers of nodes for SVM (SGD), where the solid lines with "△" markers correspond to fixed $\tau = 10$, and the dashed lines with "○" markers correspond to the proposed approach with adaptive τ.

surements on the prototype system). Our proposed approach outperforms the fixed $\tau = 10$ baseline in all cases. Furthermore, we observe that our approach scales well to a larger number of nodes for most of the cases. For Case 2, we observe a drop in accuracy when the number of nodes is larger than 10 for both the proposed approach and the fixed $\tau = 10$ baseline . This is due to the way data is distributed; when the number of nodes is larger than 10, each node observes only a single label.

We further study the instantaneous behavior of our system. Results for SVM (DGD) is shown in Fig. 2.7 for a single run of 30 seconds (for each case) on the prototype system. We see that except for Case 3 of SVM (DGD), the value of τ^* converges after a certain amount of time, showing that the control algorithm is stable. The value of τ^* keeps increasing in Case 3 of SVM (DGD) because all nodes have exactly the same data in this case and DGD uses all the data samples (i.e., no random sampling). There is no gradient deviation in this case and the optimal value of τ is infinity. As expected, the gradient deviation δ is larger for Cases 2 and 4 because the

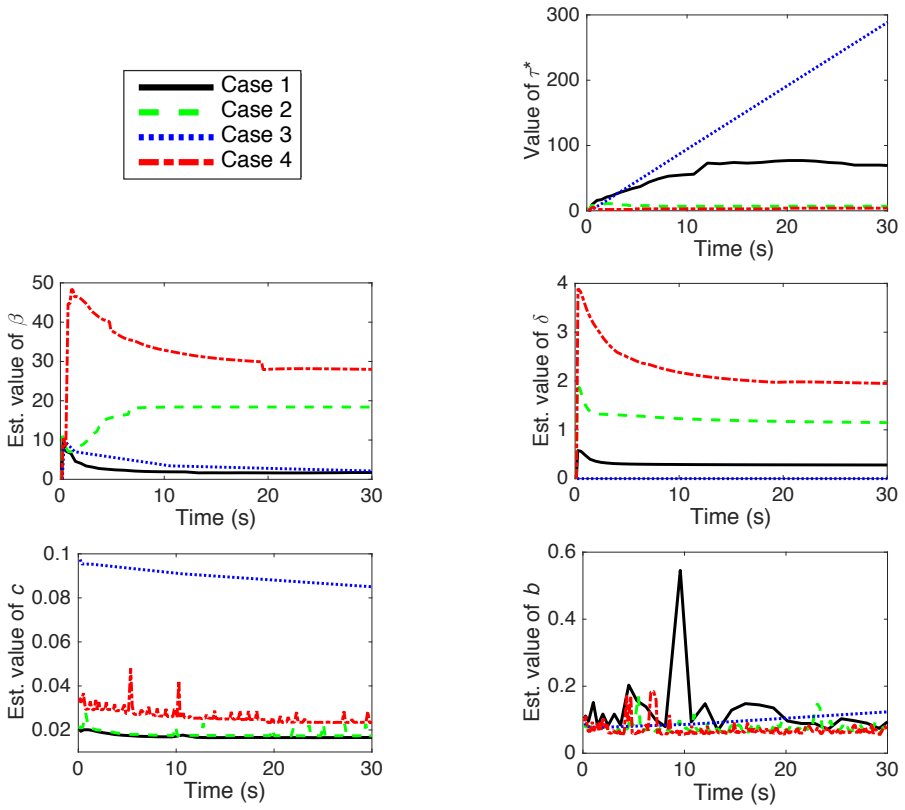


Figure 2.7: Instantaneous results of SVM (DGD) with the proposed algorithm.

data samples at different nodes are biased. The same is observed for β, indicating that the model parameter $\mathbf{w}$ is at a less smooth region for Cases 2 and 4. Case 3 of SVM (DGD) has a much larger value of c because it processes more data than in other cases, as explained before. The value of b exhibits fluctuations because of the randomness of the wireless channel. Further results of SVM (SGD) are shown in 2.8.

The sensitivity of the control parameter φ is shown in Fig. 2.9, where the experimentation settings are the same as for Fig. 2.5. We see that the relationship among τ^* in different cases is mostly maintained with different values of φ. The value of τ^* decreases approximately linearly with $\log \varphi$, which is consistent with the fact that there is an exponential term in $h(\tau)$ (and thus $G(\tau)$). For Case 3 of SVM (DGD), τ^* remains the same with different φ, because $\delta = 0$ (thus $h(\tau) = 0$) in this case and

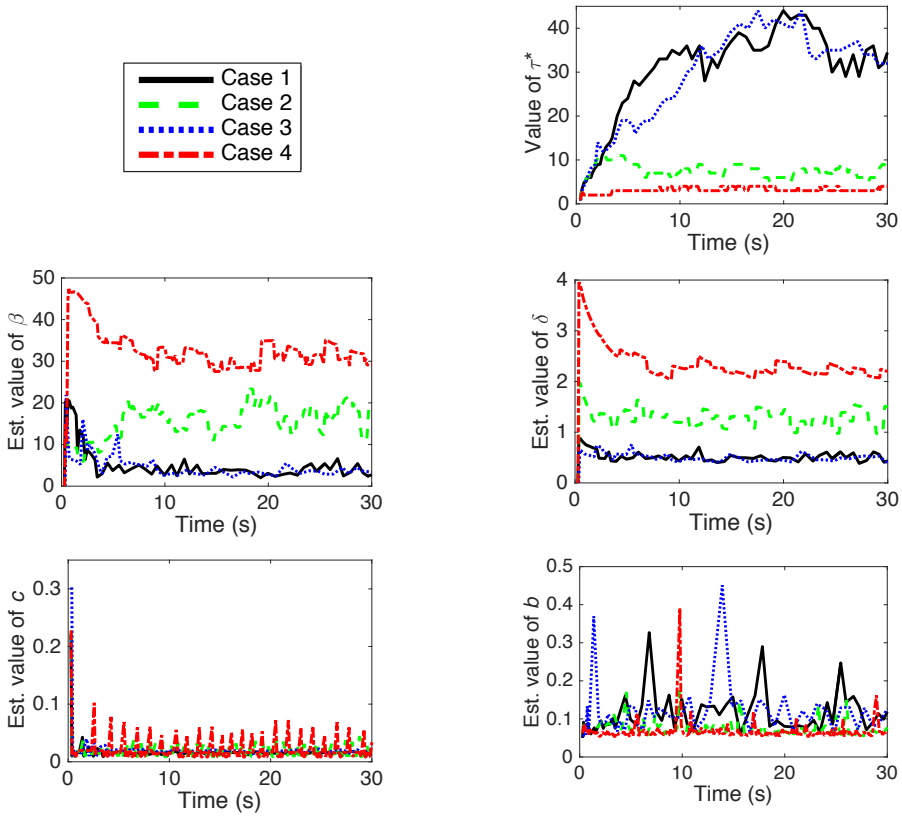
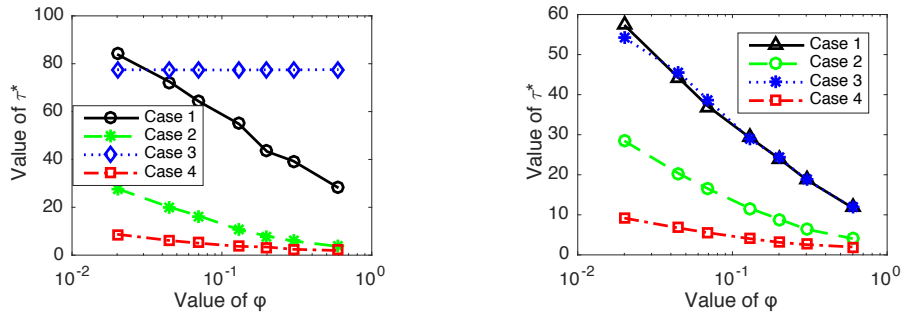


Figure 2.8: Instantaneous results of SVM (SGD) with the proposed algorithm.

the value of φ does not affect $G(\tau)$ (see (2.8)). We also see that small changes of φ does not change τ^* much, indicating that one can take big steps when tuning φ in practice and the tuning is not difficult.



(a) SVM (DGD) (b) SVM (SGD)

Figure 2.9: Impact of φ on the average value of τ^*.

2.6 Conclusion

In this chapter, we have focused on optimizing the original FedAvg algorithm under resource constraints. Using theoretical results derived in [9], a control algorithm has been proposed to achieve the desirable trade-off between local update and global aggregation in order to minimize the loss function under a resource budget constraint. Extensive experimentation results confirm the effectiveness of our proposed algorithm.

Table 2.2: Summary of main notation for Chapter 2

Notation	**Meaning**
N	number of nodes/clients
$(\mathbf{x}_j, y_j)$	data sample j
$f_j(\mathbf{w})$	individual loss function of data sample j
$F_i(\mathbf{w})$/$F(\mathbf{w})$	local loss function corresponding to node i/global loss function
$\mathcal{D}_i$/$\mathcal{D}$	local dataset at node i/ union of all local datasets
$\mathbf{w}_i$/$\mathbf{w}$	local model parameter at node i/ global model parameter
$\nabla F_i(\mathbf{w})$ /$\nabla F(\mathbf{w})$	local gradient at node i/global gradient
δ_i/ δ	Local/ Global Gradient divergence
τ	number of local updates between two aggregations
T	number of total iterations
R	total amount of resources
η	step size
ρ	parameter related to the Lipschitzeness of the loss function
β	parameter related to the smoothness of the loss function
c	unit of resource for a local update
b	unit of resource for a global aggregation
$a \triangleq \frac{b}{c}$	relative resource consumption of global aggregation
φ	Control parameter

CHAPTER 3

Efficient Federated Learning with Diverse Tasks and Data

3.1 Introduction

Federated learning approaches (including the ones discussed in Chapter 2) mostly focus on training a single model using pre-defined datasets at client devices. While such scenarios are meaningful, it can be far from being practical where each client can have a large variety of data, which may or may not be relevant to the given machine learning task. For example, if the task is to classify handwritten digits, printed digits can be considered as irrelevant data although they may be tagged with the same set of labels (thus difficult for the system to distinguish). Including irrelevant data in the federated learning process can reduce model accuracy and slow down training. In addition, there can be multiple learning tasks co-existing at the same time. For example, one task can be to train a deep learning model for classifying different animals, another task can be to train a shallow model for classifying handwritten digits, and the third task can be to learn a linear regression model for sensor measurements, etc. Considering this, a challenge is: how to support federated learning in such a realistic scenario with diverse tasks and data?

In this chapter, we consider two aspects of this challenge. The first is how does a client select relevant data for a given federated learning task ? The second aspect is, considering the co-existence of multiple tasks, how to schedule these multiple

federated learning tasks so that model training is the most efficient ?

We consider a federated learning system with a a *model requester* (MR), multiple clients and a server. A federated learning task is defined by the MR through the server. The model requester can be any user of the federated learning service and does not need to be a client involved in federated learning. The requester provides the model logic (such as a specific deep neural network architecture). The server can accept and handle multiple requests/tasks at the same time. For each request, the server first coordinates with every client to find a relevant subset of data that is available at the client, then the federated learning process starts. When there are multiple tasks, these tasks are scheduled in an appropriate manner.

Our main contributions in this chapter are as follows.

1. We propose a novel method for identifying relevant data at each client for a given federated learning task. In our proposed approach, the MR has a small set of *benchmark data* that is used as an example to capture a reasonably good input-output relationship of the trained model, but insufficient to train the model itself. The benchmark data does *not* need to be shared with the server or other clients directly, only a *benchmark model* trained on the benchmark data needs to be shared, thus preserving privacy. Using the benchmark model provided by the MR, each client identifies a subset of its data that will be involved in this federated learning task. Then, federated learning proceeds, where each client's local computation is only performed on its selected data. In this way, our approach works in a distributed manner without requiring clients or MR to share raw data.
2. We formulate the problem of scheduling multiple federated learning tasks with the goal of minimizing the completion time of every training round as a mixed-integer linear program (MILP). We show that this problem is a variant of the NP-hard flow-shop scheduling problem [3] and provide an approximate solu-

tion by relaxing and rounding the binary variables.

3. We evaluate the performance of our proposed approaches with extensive experiments. We use a variety of real-world datasets with different types of noise to show that our data selection approach outperforms other baseline approaches even when the model requester only provides a very small amount of benchmark data. By using a combination of real-world measurements and simulations, we show that our proposed scheduling mechanism is advantageous when compared to other methods.

The remainder of this chapter is organized as follows. Section 3.2 presents the related work. Section 3.3 describes the system model and definitions for federated learning. The data selection process and the scheduling algorithm are described in Sections 3.4 and 3.5, respectively. Section 3.6 presents the experimentation results and Section 3.7 draws conclusion. The summary of notations used in this chapter are in Table 3.5.

3.2 Related Work

Approaches for optimizing federated learning for resource efficiency are proposed in [18, 19, 20], which only focus on a single federated learning task. Recent work [21, 22] considers federated multi-task learning, which automatically determines whether different clients train the same or different (but related) models during the model training process. However, the tasks in [21, 22] are partitioned at the client level, i.e., each client can participate in *at most one task*, while different clients can participate in either the same or different tasks. Other work studies the case with malicious clients [23, 24, 25] and servers [26], where both defense and attacking mechanisms are considered. All the above work focuses on a unified behavior of a client or the server in its entirety (i.e., each client only participates in one federated learning

task, and the entire client/server is either malicious or not). It does not allow the more realistic settings where different subsets of data at each client can have different properties and participate in different learning tasks, which we consider in this chapter.

The case with both relevant and irrelevant data in the training dataset has been considered for centralized machine learning in recent years, where irrelevant data can be regarded as *noise* for a given machine learning task. Note that, however, noise for one learning task may be useful data for another learning task. In [27], two categories of noise are identified. Considering a supervised learning task with a fixed number of (known) classes/labels, *closed-set noise* refers to the case where a data sample in the set of known classes is labeled as another class within the set of known classes, whereas *open-set noise* is the case where a data sample *outside* the set of known classes is labeled as a class within the set of known classes[1]. In addition, *strong noise* refers to the scenario where the number of noisy data samples is arbitrary and can exceed the number of clean data samples, whereas *weak noise* refers to the case where the amount of noisy data samples is limited by a maximum number that is less than the number of clean data samples.

Among these different types of noise, approaches for training models in the weak closed-set noise setting has been most extensively studied [28, 29, 30, 31, 32], and the strong closed-set noise setting has been studied in [33, 34, 35, 36]. Very recently, the weak open-set noise setting is studied in [27, 37, 38]. We note that all the approaches that address the strong noise scenario require a small set of clean benchmark data [33, 34, 35, 36], as we do in our work. In addition, existing approaches for the open-set setting [27, 37, 38] involve very complex models that need to be trained on the entire dataset even though they only apply to the weak noise scenario, which

[1]For example, consider a multi-class classifier for classifying images of cats of dogs, an image of a dog labeled as a cat is a closed-set noise, an image of an elephant labeled as a dog is an open-set noise, because elephant does not belong to the set of known classes (cats and dogs).

makes the approaches infeasible for distributed datasets in federated learning.

In summary, there is a significant gap between the above existing work and the problem we solve in this chapter. To the best of our knowledge, none of the following aspects has been studied in the literature: 1) strong open-set noise, 2) data cleansing/filtering in federated learning with decentralized datasets, 3)the optimal scheduling of multiple federated learning tasks running simultaneously on clients. In federated learning, strong open-set noise can frequently exist in scenarios where only a small portion of the data at the client is relevant to the given task. In this chapter, we propose a novel method of filtering the data in a *distributed manner* for federated learning with *strong open-set noise*. In addition, we propose an algorithm for scheduling *multiple tasks running simultaneously* on federated learning clients.

3.3 System Model and Definitions

We consider a federated learning system with multiple clients and a server. Each client has its own local dataset with diverse types of data. As described in Section 3.1, a learning task is initiated by a model requester sending a request to the server. The model requester specifies what model needs to be trained, and also provides a small *benchmark dataset* that can be generated by the requester based on his/her own understanding of the learning task or via other means, but are deemed to have correct labels for all data in it. However, the amount of benchmark data is usually very small compared to the collection of local data at all clients and is therefore insufficient for training a highly accurate model, which is why the model requester need to make use of the federated learning collaboration.

Clients may have different amounts of relevant and irrelevant data with respect to a specific learning task. After receiving the model (training) request, the server initiates a distributed data selection process based on the benchmark data provided

by the requester, which is described in details in Section 3.4. The selected relevant subset of data is then used in the federated learning process of this task. Multiple tasks can run at the same time, and a method for scheduling these different tasks is given in Section 3.5.

In this work, we assume that the clients are trusted and cooperative. They voluntarily participate in refining the dataset for each task. The noise in the data (with respect to the task) can be due to the following reasons: 1) irrelevance of the data to the task, 2) unintentional data mislabeling by the client, 3) lack of proper representation of data labels in the system (e.g., a label "0" can represent different things depending on the context), and 4) data stored at the client is provided by an adversarial third-party that intentionally provides noisy data.

We consider a set of data $\mathcal{D}$ (potentially including noisy data samples) distributed over N clients such that $\mathcal{D} = \bigcup_{n=1}^{N} \mathcal{D}_n$, where $\mathcal{D}_n$ is the dataset located at client n. We focus on supervised learning with labeled training data in this work. The *loss function* of a labeled data sample $(x_i, y_i) \in \mathcal{D}$ is defined as $l(f(x_i, \theta), y_i)$, where x_i is the input to the model, y_i is the desired output (label) of the model, $f(x_i, \theta) = \hat{y}_i$ is the predicted output of the model, and θ is the model parameter vector. The function $f(x_i, \theta)$ captures the model logic and can be different for different models. The loss function $l(f(x_i, \theta), y_i)$ is an error function in the form of mean squared error, cross entropy, etc.

For a federated learning task, let $\mathcal{F} \subseteq \mathcal{D}$ denote the set of samples that is selected as relevant to the given task. This set is found by finding the subset $\mathcal{F}_n \subseteq \mathcal{D}_n$ in each client (node) n with $\mathcal{F} = \bigcup_{n=1}^{N} \mathcal{F}_n$ and the details of this selection process will be given in Section 3.4. The overall loss of relevant data at client n is defined as $L_n(\theta) = \frac{1}{|\mathcal{F}_n|} \sum_{(x_i, y_i) \in \mathcal{F}_n} l(f(x_i, \theta), y_i)$, where $|\cdot|$ denotes the cardinality of the set, based on which the global loss across all clients is defined as $L(\theta) = \frac{\sum_{n=1}^{N} |\mathcal{F}_n| L_n(\theta)}{\sum_{n=1}^{N} |\mathcal{F}_n|}$. The goal of federated learning on the selected data subset $\mathcal{F}$ is to find the model

parameter vector $\hat{\theta}$ that minimizes $L(\theta)$:

$$\hat{\theta} = \arg\min_{\theta} L(\theta). \tag{3.1}$$

The minimization problem in (3.1) is solved in a distributed manner using standard federated learning procedure [2, 18] as described in Chapter 2.

3.4 Data Selection

Data selection is performed separately for each federated learning (model training) task. The goal is to filter out noisy data samples for the specific learning task. The filtering process is decentralized, where each client performs the filtering operation on its own dataset and only exchanges a small amount of meta-information with the server.

The main idea is that the server first trains a *benchmark model* (of the same type as the model requested by the requester) using only a subset of the benchmark data provided by the requester. To determine the relevance, the loss $l(f(x_i, \theta), y_i)$ is evaluated for each data sample $(x_i, y_i) \in \mathcal{D}$ using the benchmark model. By comparing the distribution of loss values of data samples at the clients and a held-out set of the benchmark data (that is not used for training the benchmark model), we determine a threshold for the loss $l(f(x_i, \theta), y_i)$; data samples with a loss higher than the threshold are regarded as noise and excluded from this federated learning task.

Fig. 3.1 illustrates the overall data selection process done by the server and clients, which includes the following steps:

- The server builds (trains) the benchmark model using a small benchmark dataset without noise (Steps 1 and 2).

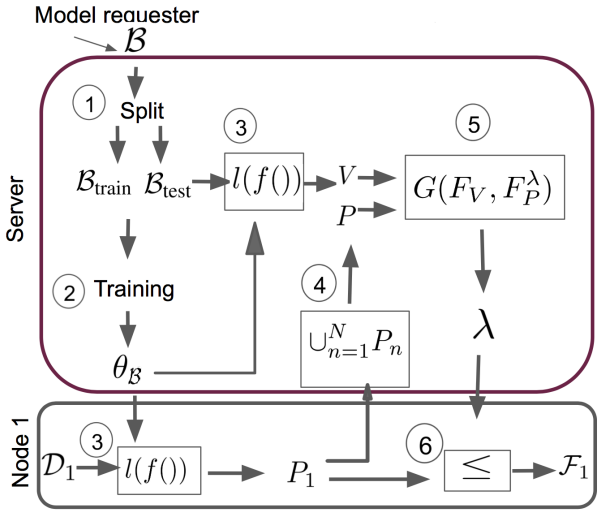


Figure 3.1: Data selection procedure.

- Each client evaluates its own dataset against the benchmark model, and creates a list of loss values from the benchmark model of all of its data samples (Step 3).
- The server merges the lists of the loss values from all clients, and compares the distribution of these loss values against that of loss values of the benchmark dataset, to calculate the filtering threshold (Steps 4 and 5).
- Each client filters out noisy data samples in its dataset using the filtering threshold (Step 6).

Note that this process is performed *without transmitting raw data* from the clients to the server; only a benchmark model and the losses are exchanged, hence suitable for federated learning settings. In the following, we provide the detailed description of this filtering process by formally establishing its objective.

3.4.1 Objective

Assume that a model requester wants to train a classifier able to recognize the classes (labels) defined in a label set C and provides a set of benchmark data $\mathcal{B}$, which

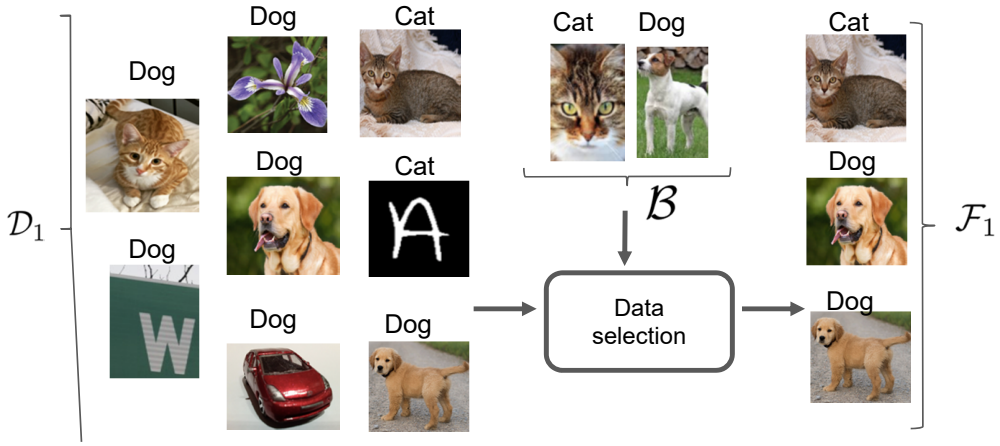


Figure 3.2: Example of noisy data at clients ($\mathcal{D}_1$), benchmark dataset ($\mathcal{B}$), and the filtered dataset ($\mathcal{F}_1$).

contains only a small number of data samples correctly labelled from each of the categories defined in C. Typically, $|\mathcal{B}| \ll |\mathcal{D}|$. For example, a model requester focusing on recognizing images of *cats* and *dogs* will provide a few correctly labeled examples representing *cats* and *dogs*, and $C = \{\text{dog}, \text{cat}\}$ is implicitly defined.

Our overall goal of data selection is to find the subset $\mathcal{F} \subseteq \mathcal{D}$ where $\mathcal{F}$ is the union of subset $\mathcal{F}_n$ identified at each client n, (i.e. $\mathcal{F} = \bigcup_{n=1}^{N} \mathcal{F}_n$), such that the *testing* loss function evaluated on the classes defined by C is minimized:

$$\min_{\mathcal{F}} \quad \sum_{(x_i,y_i)\in\mathcal{T}} l(f(x_i,\hat{\theta}); y_i) \tag{3.2}$$

where the test dataset $\mathcal{T} = \{(x_i, y_i) : y_i \in C\}$ is a held-out dataset, separated from the training dataset $\mathcal{D}$ and the benchmark dataset $\mathcal{B}$ (i.e., $\mathcal{T} \cap \mathcal{D} = \emptyset$ and $\mathcal{T} \cap \mathcal{B} = \emptyset$), and $\hat{\theta}$ is the parameter vector of the model obtained from (3.1).

Fig. 3.2 shows an example of a noisy dataset $\mathcal{D}_1$, a benchmark dataset $\mathcal{B}$ provided by the model requester, and the ideal subset $\mathcal{F}_1$, obtained using our proposed filtering method.

3.4.2 Training Benchmark Model

First, the benchmark dataset $\mathcal{B}$ is divided into a training set $\mathcal{B}_{\text{train}}$ and a test set $\mathcal{B}_{\text{test}}$ (Step 1 in Fig. 3.1) such that $\mathcal{B}_{\text{train}} \cap \mathcal{B}_{\text{test}} = \emptyset$. Then, the benchmark model is trained using $\mathcal{B}_{\text{train}}$, whose parameter $\theta_{\mathcal{B}}$ is obtained by minimizing the following expression using a model training process, such as the stochastic gradient descent (Step 2):

$$\theta_{\mathcal{B}} = \arg\min_{\theta} \quad \frac{1}{|\mathcal{B}_{\text{train}}|} \sum_{(x_i,y_i)\in\mathcal{B}_{\text{train}}} l(f(x_i,\theta), y_i) \ . \tag{3.3}$$

3.4.3 Finding Loss Distribution Using Benchmark Model

The model parameter $\theta_{\mathcal{B}}$ is used to find two sets of loss values (Fig. 3.1, Step 3), evaluated on $\mathcal{B}_{\text{test}}$ and $\mathcal{D}_n$ respectively:

$$V = \{l(f(x_i,\theta_{\mathcal{B}}), y_i) : \forall (x_i, y_i) \in \mathcal{B}_{\text{test}}\} \tag{3.4}$$

$$P_n = \{l(f(x_i,\theta_{\mathcal{B}}), y_i) : \forall (x_i, y_i) \in \mathcal{D}_n\}. \tag{3.5}$$

Note that V is obtained by the server from $\mathcal{B}_{\text{test}}$, and each P_n is obtained from $\mathcal{D}_n$ by each client n. Then all P_n's from the clients are sent to the server and combined to obtain the loss value set of all clients' data (Step 4), i.e., $P = \bigcup_{n=1}^{N} P_n$.

The set V provides a reference distribution of loss values, against which the relevance of the data samples at each client n is assessed. Intuitively, we assume that the smaller the individual loss value $l(f(x_i,\theta_{\mathcal{B}}), y_i)$ is, the more (x_i, y_i) will fit to the benchmark model defined by $\theta_{\mathcal{B}}$, hence being more likely relevant to the learning task under consideration.

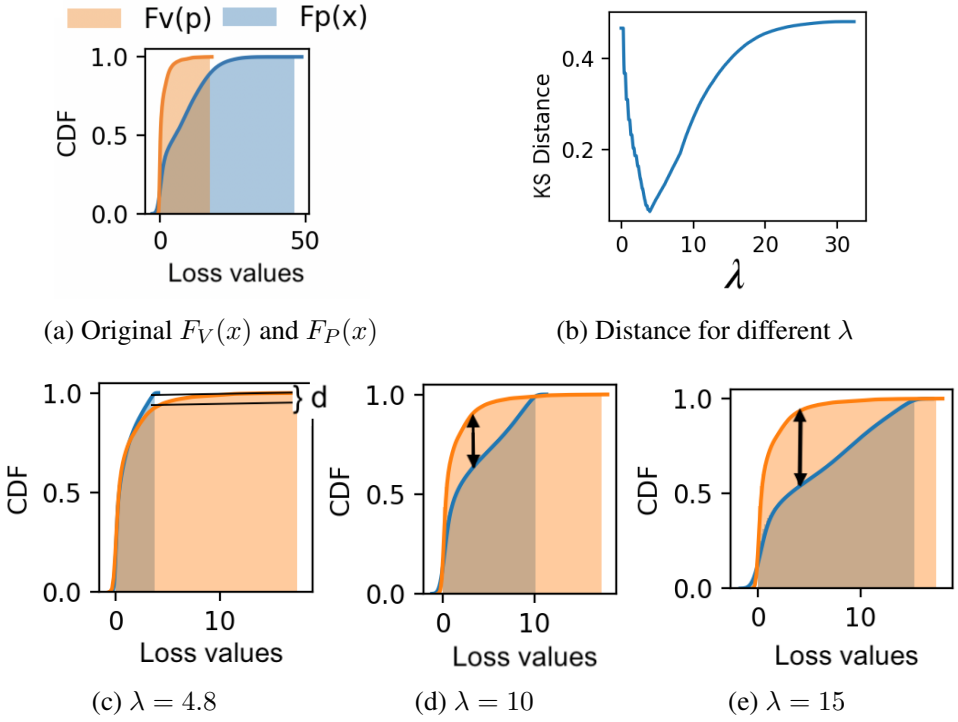
3.4.4 Calculating Filtering Threshold in Loss Values

We use V as a benchmark distribution of acceptable range of individual loss values for data samples in $\mathcal{D}$. In other words, we assume that data samples, whose loss values evaluated with the benchmark model are within an "acceptable" range in the distribution of V, have a high probability to be relevant to the learning of the target model. Inversely, if a sample has a loss value out of this acceptable range, there is a high probability that this sample will either corrupt the model training or just be irrelevant.

The goal is then to find a good threshold in the loss value to determine which data samples to include in the set of relevant (not noisy) data $\mathcal{F}_n$. Note that this can be seen as an outlier detection, where the outliers are defined as the irrelevant data samples with respect to the target model, and their detection is performed in the 1-dimensional space of loss values mapped from the original data space via the benchmark model. Note also that we have defined V as the set of loss function values evaluated on $\mathcal{B}_{\text{test}}$ and not $\mathcal{B}_{\text{train}}$, in order to avoid loss values over-fitted to the training dataset.

Our approach to detecting the outliers (i.e., noisy data) is to use V as a mask to find an upper limit of acceptable loss values via a statistical test that compares the distribution of V and P. More specifically, let us denote the empirical Cumulative Distribution Function (CDF) of V and P by F_V and F_P, respectively; that is, $F_V(x) = \Pr\{X \leq x : X \in V\}$ and $F_P(x) = \Pr\{X \leq x : X \in P\}$. We further denote by F_P^λ the conditional CDF of P such that $F_P^\lambda(x) = \Pr\{X \leq x | X \leq \lambda : X \in P\}$. Note that we call this conditional CDF the "truncated" CDF as the maximum range of values is truncated to λ, i.e., $F_P^\lambda(x) = 1$ if $x \geq \lambda$, and $F_P^\lambda(x) = F_P(x)/F_P(\lambda)$ if $x < \lambda$.

Given λ, we define the distance G between the two distributions F_V and F_P^λ by the Kolmogorov-Smirnov (KS) distance [39], which is often used to quantify the



(a) Original $F_V(x)$ and $F_P(x)$

(b) Distance for different λ

(c) $\lambda = 4.8$

(d) $\lambda = 10$

(e) $\lambda = 15$

Figure 3.3: KS distance computation and optimal λ for $F_V(x)$ and $F_P^\lambda(x)$).

distance between two CDFs. Specifically, $G(F_V, F_P^\lambda) = \sup_x |F_V(x) - F_P^\lambda(x)|$.

Then, we calculate our threshold in loss values, denoted by λ^*, that minimizes G. That is:

$$\lambda^* = \arg\min_\lambda G(F_V, F_P^\lambda). \tag{3.6}$$

Fig. 3.3 illustrates this process with an example. Given the unconditional CDFs $F_V(x)$ and $F_P(x)$ in Fig. 3.3a, Figs. 3.3c, 3.3d, and 3.3e show the KS distance between $F_V(x)$ and the truncated CDF $F_P^\lambda(x)$ for different values of λ. In this example, $\lambda^* \approx 4.8$ because it minimizes the KS distance between F_V and F_P^λ as shown in Fig. 3.3b.

3.4.5 Local Selection of Data by Clients

After computing λ^* according to (3.6), the server sends λ^* to all the clients. Then, each client makes the selection of relevant data locally (Step 6 in Fig. 3.1):

$$\mathcal{F}_n = \{(x_i, y_i) \in \mathcal{D}_n : l(f(x_i, \theta_{\mathcal{B}}), y_i) \leq \lambda^*\}. \tag{3.7}$$

Once the selection is made locally for every client, the standard federated learning process starts, where each client n performs stochastic gradient descent on the selected data $\mathcal{F}_n$. As explained in Section 3.3, the mini-batch size is adapted according to the size of $\mathcal{F}_n$. In the extreme case where a client has no data of interest to a model requester, the mini-batch size for this client will be 0 and this client is excluded from the federated learning task under consideration.

3.5 Scheduling of Federated Learning Tasks

When multiple federated learning tasks co-exist in the system, the optimal scheduling of these tasks is necessary to maximize the system efficiency. A naive first-come-first-served based approach to task allocation is not suitable in this setting, because some small models may train much faster than other large models, and it is not reasonable for a model that can be trained within a few minutes to wait for a model that needs to be trained for several weeks to complete, for instance. Considering this, we focus on a scheduling mechanism in this work that includes multiple *cycles*. In each cycle, we run *every* federated learning task that is currently in the system for τ gradient descent steps of local iteration, before starting the next cycle.

3.5.1 Definitions

Assume that the system currently has K tasks. We consider a cycle in the system as the duration for all the clients $n = 1, 2, ..., N$ to *download* the model parameter vector $\theta_n^{(k)}(t)$, *compute* τ steps of model update according to (2.4) on filtered local data, and *upload* the new $\theta_n^{(k)}(t+\tau)$ to the server, for all $k = 1, 2, ..., K$ tasks (models). Our goal of scheduling is to *minimize the time of a cycle*. We assume that the communication link between the server and each client is independent from each other, which is a reasonable assumption because the access network bandwidth of each client is usually much lower than the bandwidth available to the server. Thus, minimizing the time of a cycle involving all the clients is equivalent to minimizing the time of a cycle for each client, and we focus on a single client in the following.

We assume that each client n has a separate downlink and uplink for communication, as many home Internet connections nowadays have separate downlink and uplink that are not mixed together. Our formulation can be easily modified to the case with shared downlink and uplink as well, but we focus on separate downlink and uplink in this work as it is the more challenging case. Considering a client n and a given model k, the amount of time to download, compute, and upload model k is denoted by d_k, c_k, and u_k, respectively, where we omit the subscript n for simplicity since we focus on a single client, as explained above. A client can download, compute, and upload only one model at a time. For a model k, the *precedence relationship* is required such that download has to finish before compute, and compute has to finish before upload. However, downloading a model k can happen at the same time with computing another model k', where $k \neq k'$. Different models can have different download and upload durations (d_k and u_k) depending on the sizes of their parameter vector $\theta_n^{(k)}$, and different computation times c_k depending on their relevant data $\mathcal{F}_n^{(k)}$, which determines the mini-batch size (see Section 3.3). Thus, it is useful to determine the optimal sequence of processing the K tasks at the given

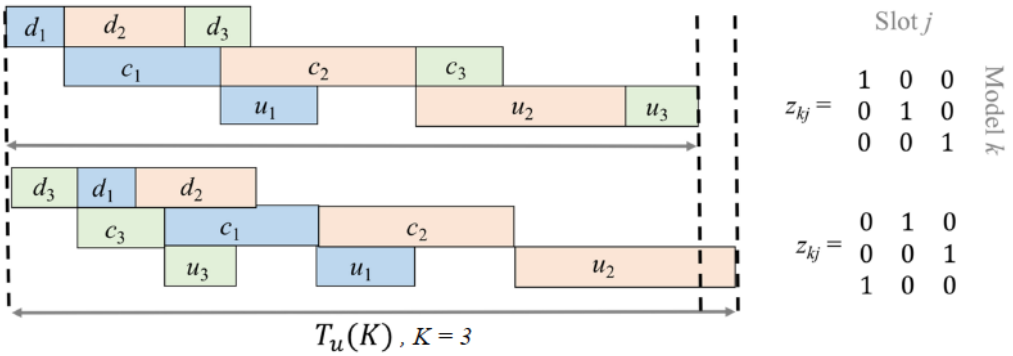


Figure 3.4: An example of two possible sequences to schedule three models. On the top, model 1 is scheduled first, model 2 second, and model 3 at the end. On the bottom, model 3 is scheduled first, then model 1, and model 2 at last. The first scheduling sequence at the top is better than the second sequence, as the overall completion time $T_u(K)$ with $K = 3$ of the first sequence is smaller.

client n, as different processing sequences yield different completion times of the cycle (see Fig. 3.4). Note that since the communication links between the server and different clients are assumed to be independent, as mentioned earlier, the download and upload of the parameter vector $\theta_n^{(k)}$ for a given model k can occur at different times for different clients.

3.5.2 Problem formulation

With the above setting, our problem is an instance of the flow-shop scheduling problem, where it is known that the optimal solution gives the same sequence of models in the download, compute, and upload stages (i.e., downloading model 1 before model 2 but computing model 2 before model 1 is not beneficial) [40, Lemma 1]. For this reason, we can consider K *slots* $j = 1, 2, ..., K$ to represent the scheduling sequence of models at some given client n. We define the binary variable z_{kj}, which is equal to one if model k is scheduled in slot j, i.e., model k is processed (downloaded, computed, and uploaded) at the j-th place within a cycle, and zero otherwise. A model can be assigned to only one slot and a slot can only have one model assigned to it, i.e., $\sum_k z_{kj} = 1 \ \forall j$ and $\sum_j z_{kj} = 1 \ \forall k$. Note that the slot

assignment only defines the scheduling sequence of models; the actual completion time of download/compute/upload for the model in the j-th slot is computed according to

$$T_d(j) = T_d(j-1) + \sum_{k=1}^{K} d_k z_{kj} \tag{3.8}$$

$$T_c(j) = \max\{T_d(j), T_c(j-1)\} + \sum_{k=1}^{K} c_k z_{kj} \tag{3.9}$$

$$T_u(j) = \max\{T_c(j), T_u(j-1)\} + \sum_{k=1}^{K} u_k z_{kj} \tag{3.10}$$

where $T_d(j)$, $T_c(j)$, and $T_u(j)$ are the duration of the cycle starting from the beginning until completion of the downloading, computing, and uploading respectively for the model that has been scheduled in the j-th slot. The maximum operators in (3.9) and (3.10) capture the precedence relationship among download, compute, and upload. We define $T_d(0) = T_c(0) = T_u(0) = 0$.

With this definition, the overall completion time of the cycle is $T_u(K)$, i.e., the time that the last model has been uploaded, as shown in Fig. 3.4. Different clients may have different values of $T_u(K)$ and the overall completion time of the cycle with all clients is the maximum $T_u(K)$ among all clients. We wait for the completion of all clients before starting the next cycle, because otherwise we will have asynchronous updates from clients, which is harmful for federated learning with non-i.i.d. data [18]. By minimizing $T_u(K)$ for each client, we also minimize the maximum $T_u(K)$ among all clients.

The minimization problem of $T_u(K)$ for each client can be formulated as an MILP as follows:

$$\min_{\{z_{kj}\}} \quad T_u(K) \tag{3.11a}$$

$$\text{s.t. } T_d(j) = T_d(j-1) + \sum_{k} d_k z_{kj}, \qquad \forall j \in \{1, ..., K\} \tag{3.11b}$$

$$T_c(j) \geq T_c(j-1) + \sum_k c_k z_{kj}, \qquad \forall j \in \{1, ..., K\} \tag{3.11c}$$

$$T_u(j) \geq T_u(j-1) + \sum_k u_k z_{kj}, \qquad \forall j \in \{1, ..., K\} \tag{3.11d}$$

$$T_c(j) \geq T_d(j) + \sum_k c_k z_{kj}, \qquad \forall j \in \{1, ..., K\} \tag{3.11e}$$

$$T_u(j) \geq T_c(j) + \sum_k u_k z_{kj}, \qquad \forall j \in \{1, ..., K\} \tag{3.11f}$$

$$\sum_k z_{kj} = 1, \qquad \forall j \in \{1, ..., K\} \tag{3.11g}$$

$$\sum_j z_{kj} = 1, \qquad \forall k \in \{1, ..., K\} \tag{3.11h}$$

$$T_d(j), T_c(j), T_u(j) \geq 0, \qquad \forall j \in \{0, ..., K\} \tag{3.11i}$$

$$z_{kj} \in \{0, 1\}, \qquad \forall j, k \in \{1, ..., K\} \tag{3.11j}$$

where (3.11b) is the same as (3.8). Constraints (3.11c)–(3.11f) are equivalent to (3.9) and (3.10) but without maximum operator, since the objective of (3.11) is a minimization. Finally, (3.11g) and (3.11h) ensure that each model is assigned to one and only one slot.

3.5.3 Hardness

Theorem 1. *The problem (3.11) is NP-hard, even if $d_k = \gamma u_k$ for all k, for some constant $\gamma > 0$.*

Proof. For arbitrary values of d_k and u_k, the result is the same as [3, Theorem 1], which shows that the flow-shop scheduling problem with three machines is NP-hard. When $d_k = \gamma u_k$, the NP-hardness can be shown using a similar procedure as the proof of [3, Theorem 1] by reduction from the NP-complete 3-partition problem.

The 3-partition problem [3] asks: given integers $M > 0$, $H > 0$, and a set of integers $\mathcal{A} = \{a_1, a_2, ..., a_{3M}\}$, with $\sum_{m=1}^{3M} a_m = MH$ and $\frac{H}{4} < a_m < \frac{H}{2}$ ($\forall m =$

$\{1, 2, ..., 3M\}$), does there exist a partition of $\mathcal{A}$ into 3-element sets $A_1, ..., A_M$ such that $\sum_{a \in A_m} a = H$, for all $m = \{1, 2, ..., M\}$?

We construct an instance of our problem with $K = 4M + 3$ models in total, out of which $M + 1$ models have $d_k = u_k = 2H$ and $c_k = H$, $3M$ models have $d_k = u_k = 0$ and $c_k = a_m$ for $m = \{1, 2, ..., 3M\}$, and two models have $d_k = u_k = 0$ and $c_k = 2H$. Using a similar argument as in the proof of [3, Theorem 1], we can show that if the optimal solution to our problem (3.11) gives $T_u(K) \leq (2M + 5)H$, then the answer to the 3-partition problem is "yes". The reverse is also true. Since the 3-partition problem is NP-complete, our problem is NP-hard. □

We specifically consider the case of $d_k = \gamma u_k$ above, because for a given downlink and uplink bandwidth, the ratio $\frac{d_k}{u_k}$ is always a constant (denoted by γ) for any k, since the size of the parameter vector of any model k remains the same in the downlink and uplink.

3.5.4 Algorithm

We present a simple algorithm for approximately solving (3.11) in the following. First, we relax the binary constraint (3.11j) to $0 \leq z_{kj} \leq 1$ ($\forall j, k \in \{1, ..., K\}$), so that the resulting problem becomes a linear program (LP) that can be efficiently solved in polynomial time using existing solvers. Then, we sort the values of z_{kj} in descending order, assign $z_{kj} = 1$ starting from the largest value of z_{kj} if doing so does not violate the constraints (3.11g) and (3.11h). The algorithm is given in Algorithm 1. It can be easily seen that the complexity of this algorithm is bounded by the complexity of the LP solver. When ignoring the LP solver's complexity, the sorting in line 2 has $O(K^2 \log K)$ complexity for K^2 values. The initialization in line 3 has $O(K^2)$ complexity. The assignment in lines 4–7 has $O(K^3)$ complexity where we note that line 6 has $O(K)$ complexity. Hence, the complexity of the

Algorithm 1: Algorithm for approximately solving (3.11)

Input: $\{d_k\}, \{c_k\}, \{u_k\}$ for all $k = 1, 2, ..., K$
Output: $\{z_{kj}\}$

```
Obtain continuous values {z'_kj} of the relaxed problem from LP solver;
Sort {z'_kj} in descending order (ties are arbitrarily broken), let (k, j) = Z(p) denote the
  p-th largest value of {z'_kj};
Set z_kj ← 0 for all j, k ∈ {1, ..., K};
for p = 1, 2, ..., K^2 do
    (k', j') ← Z(p);
    if not (∃j such that z_k'j = 1 or ∃k such that z_kj' = 1) then
        z_kj ← 1;
```

rounding procedure (not including LP solver) in Algorithm 1 has a time complexity of $O(K^2 \log K + K^2 + K^3) = O(K^3)$.

3.6 Experimentation Results

3.6.1 Set up

We conduct experiments in a federated learning system with a large number of simulated clients ($N \geq 100$). The scheduling algorithm is further evaluated based on measurements on real Raspberry Pi Version 4 devices as clients.

3.6.1.1 Datasets

We use seven image datasets as listed in Table 3.1, which can be grouped into three categories. The first two datasets represent digits (0–9): MNIST [14] for handwritten digits and SVHN [41] for street-view house numbers. The next two datasets contain images of English characters and digits ('a'-'z', 'A'-'Z', and '0'-'9'): FEMNIST [42] for handwritten ones, and Chars74K [43] for a mix of characters obtained from outdoor images, hand-written characters, and computer fonts. The last three datasets represent images of various objects: FASHION [44] for fashion items, and CIFAR-10 and CIFAR-100 [45] for different types of objects (e.g., vehicles, animals).

3.6.1.2 Data Partition Among Clients

We partition data into clients in a non-i.i.d. manner as in realistic federated learning scenarios. For the FEMNIST dataset, which is already partitioned by the writers, we consider the images from each writer as belonging to a separate client. For all other datasets, the data are distributed into clients so that each client has only one class (label) of data from that dataset. The clients and labels are mapped randomly, in a way that the number of clients for each label differ by at most one. For all the clients with the same label of data, the data with this label are partitioned into clients randomly. When multiple datasets are mixed together (the open-set noise setting), different clients have different proportions of samples from each dataset, resulting in different amount of data samples in total. We have $N = 370$ clients in any experiment involving FEMNIST, which is obtained using the default value in FEMNIST for non-i.i.d. partition[2]. For experiments without FEMNIST, we assume $N = 100$ clients.

3.6.1.3 Noisy Data

Open-set noisy data (i.e., case where a data sample *outside* the set of known classes is labeled as a class within the set of known classes (See section 3.2)), are constructed by adding data from other datasets as noise to a given "target" dataset (i.e., the clean dataset for the model being trained in the task). We mix the noise data samples to the target dataset, and preserve their labels in the mixed dataset[3]. For example, we sample some data from SVHN (acting as noise, with labels 0–9) and add them to MNIST (acting as the target dataset, with labels 0–9) while keeping the same label, e.g., data with label 0 in SVHN is mixed only with data with label 0 in MNIST. Even if the number of labels in the noise dataset is different from that of the target dataset,

[2] https://github.com/TalwalkarLab/leaf/tree/master/data/femnist

[3] This is a common practice for simulating open-set noise [27, 36].

we apply the noise only to the labels that are common to both datasets; e.g., CIFAR-10's data with 10 labels are added to the corresponding first 10 classes of FEMNIST, or CIFAR-100's first 62 classes of data are mixed to the corresponding classes of FEMNIST. Additionally, in order to mix different datasets and train them together, we transform the open-set noise data such that their dimensions are the same as that of the target dataset (e.g., when training a classifier for SVHN with MNIST as noise, we transform MNIST data to color image, and resize it to 32×32 pixels).

In closed-set noise (i.e., case where a data sample in the set of known classes is labeled as another class within the set of known classes (See section 3.2)) settings where the noise and the clean data both belong to a single dataset, which can be seen as a special case of the open-set noise setting, a subset of the dataset is mislabeled from one class to another.

3.6.1.4 Benchmark Data

The benchmark dataset used to build the benchmark model is obtained by sampling a certain percentage of data from the original (clean) datasets. For all datasets other than FEMNIST, the benchmark dataset is sampled randomly from the training data. For FEMNIST, which is pre-partitioned, the benchmark dataset only includes data from a small subset of partitions (clients).

3.6.1.5 Model and Baseline Methods

We use a convolutional neural network (CNN) as the classifier for all experiments. The CNN architecture is the same as the one used in [18]. We train the CNN model using stochastic gradient descent (SGD) with $\eta = 0.01$, $\tau = 10$, and a mini-batch size of 8% of the selected data $\mathcal{F}_n$ at each client n (see Section 3.3).

To evaluate our data selection method, we consider three baseline methods for comparison: (i) model trained only on the small benchmark dataset (referred to as

Table 3.1: Summary of Datasets used in Experiments

Dataset	# training	# testing	Category	Format
MNIST[14]	60,000	10,000	10	$28 \times 28 \times 1$
SVHN[41]	73,257	26,032	10	$32 \times 32 \times 3$
FEMNIST[42]	71,090	8,085	62	$28 \times 28 \times 1$
Chars74K[43]	58,097	17,398	62	$28 \times 28 \times 1$
FASHION [44]	60,000	10,000	10	$28 \times 28 \times 1$
CIFAR-10 [45]	50,000	10,000	10	$32 \times 32 \times 3$
CIFAR-100 [45]	50,000	10,000	100	$32 \times 32 \times 3$

	Description
MNIST[14]	hand-written digits
SVHN[41]	cropped digits from street view
FEMNIST[42]	hand-written characters (federated learning settings)
Chars74K[43]	natural images, hand-written and fonts characters
FASHION [44]	fashion items
CIFAR-10 [45]	various object classes
CIFAR-100 [45]	various object classes

the *benchmark model*), (ii) model trained using only the target dataset without noise, and (iii) model trained with all the (noisy) data at clients without data selection.

To evaluate the scheduling method, the following baseline approaches are considered: (i) *LP-relax (lower bound)* which is the result of the relaxed problem of (3.11) with continuous values of z_{kj} (this gives a lower bound of the original problem), (ii) *LP-relax + rounding* obtained by our Algorithm 1, (iii) *Random* obtained by randomly selecting binary values of $z_{k,j}$ such that the constraints (3.11g) and (3.11h) hold, and (iv) *Round-Robin* where one model is downloaded, computed (updated), and uploaded, before the download of the next model starts. For some experiments, we also compute the *Optimal* result of (3.11) with binary $\{z_{kj}\}$. The LP and MILP problems are solved using the PuLP solver[4].

[4] https://pythonhosted.org/PuLP/solvers.html

3.6.2 Results

3.6.2.1 Data Selection Performance with Different Types of Noise

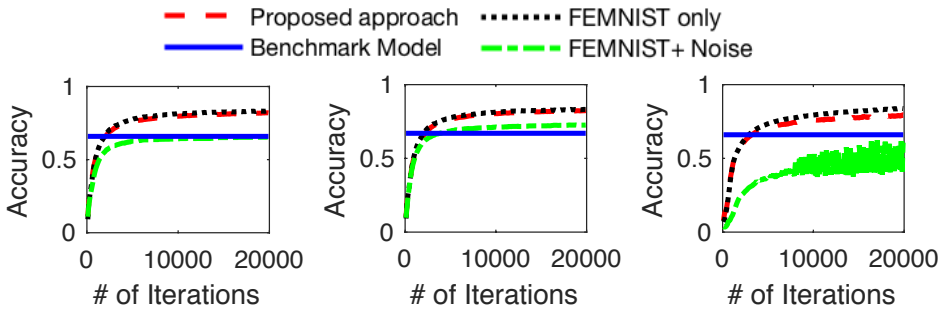
We first conduct experiments with different types of noise: (i) open-set noise from datasets in the same category, (ii) open-set noise from a different category, and (iii) closed-set noise in the same dataset. In particular, we use FEMNIST as the target dataset, and mix Chars74k into it as the same-category open-set noise, and CIFAR-100 as the different-category open-set noise[5]. In both cases, we produce the noisy data in a 1:1 target to noise ratio in terms of the number of samples. For the closed-set noise, we mislabel samples that have labels from 0 to 31 by adding +2 to the true label, in 75% of the clients, resulting in approximately 37.5% (i.e. half of 75% of the data) of the total dataset being mislabelled. The benchmark dataset is generated by sampling up to 5% of FEMNIST (see Section 3.6.1.4).

Fig. 3.5 shows the accuracy achieved by the four models (obtained by our method and the three baselines) on the test data when they are trained with the above three noisy data settings. The performance of the benchmark model is shown as a constant in Fig. 3.5 (and also in Fig. 3.7 later), because it is trained at the server before federated learning starts. We see that, in all cases, our approach always performs very close to the best case baseline ("FEMINIST only"), and significantly better than the benchmark model and the one trained with the entire noisy dataset. This shows the robustness of our approach to both open-set and closed-set noises.

Fig. 3.6 shows the results of repeating the same experiments but with the amount of benchmark data varied from 1% to 5% of the original dataset. The results, shown as the accuracy on the test data when the model training has converged, clearly indicate that the model built with our data selection performs very close to the best-case baseline, while outperforming the other two baselines (benchmark model and noisy

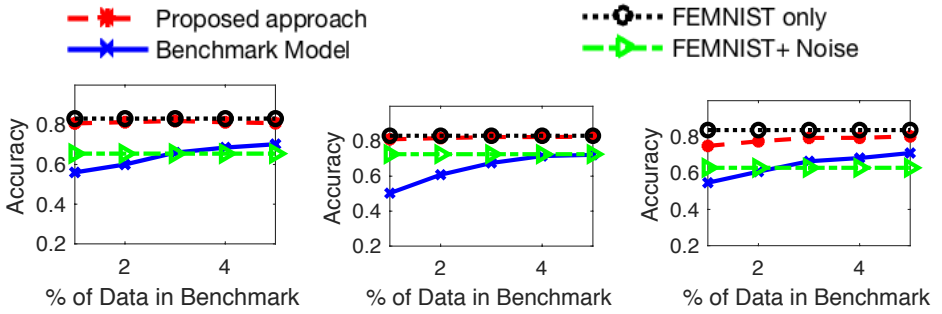
[5]We observe similar results from other combinations of target and noise datasets, which are omitted in this work for brevity.

data model), for all sizes of the benchmark dataset. We also see that the performance of the benchmark model improves with the (clean) benchmark dataset size as expected, while the performance of our approach remains nearly constant. This shows that our approach works well even with a very small amount of benchmark data (such as 1% of the original data).



(a) Noise: CHAR-74 (b) Noise: CIFAR-100 (c) Noise: Closed-set

Figure 3.5: Classifying FEMNIST under different types of noise, when the amount of benchmark data is 3% of the original data.



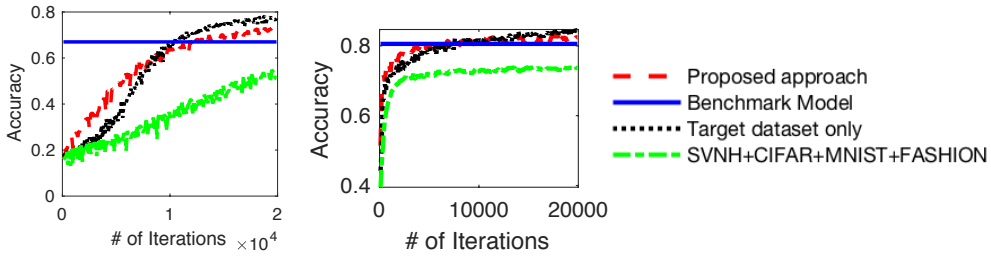
(a) Noise: CHAR-74, Target: FEMNIST (b) Noise: CIFAR-100, Target: FEMNIST (c) Noise: Closed-set, Target: FEMNIST

Figure 3.6: Varying size of benchmark data when classifying FEMNIST under different types of noise.

3.6.2.2 Data Selection Performance (Strong Noise)

We then conduct experiments to assess our data selection method when we further increase the level of the noise such that 75% of the training data are noise. Fig. 3.7 shows the testing accuracy results for the cases when (i) SVHN is the target dataset

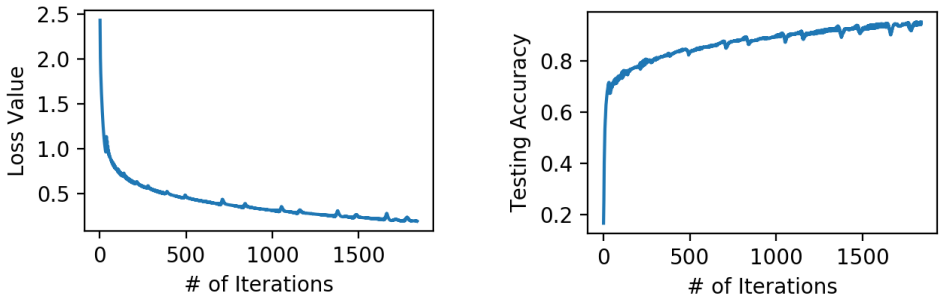
with CIFAR-10, MNIST, and FASHION as noises (Fig. 3.7a), and (ii) FASHION is the target dataset with CIFAR-10, SVHN, and MNIST as noises (Fig. 3.7b). The overall trend in the performance is similar to what are observed with mild noise levels in Figs. 3.5. Our approach achieves a model accuracy very close to the best-case baseline, while significantly outperforming the case without data selection.



(a) Target: SVHN (b) Target: FASHION

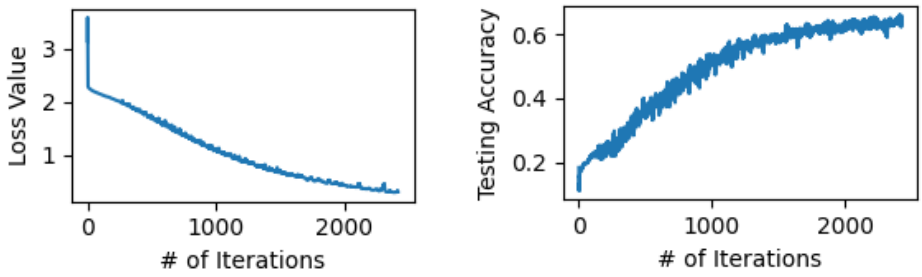
Figure 3.7: Strong open-set noise scenario, when the amount of benchmark data is 3% of the original data.

The benchmark model in this case performs reasonably well, too, since a relatively small amount of training data for these two target datasets (SVHN and FASHION) is generally sufficient to train a model achieving good accuracy. Note that the performance of the benchmark model is shown as a constant in the figures above, because it is trained at the server before federated learning starts. Fig. 3.8 and Fig. 3.9 show the loss value and the testing accuracy when training the benchmark model with 3% of FASHION and SVHN respectively. Note that the number of iterations for the benchmark model differs for FASHION and SVHN, this is because we stop training when reaching convergence.



(a) Training Loss (b) Testing Accuracy

Figure 3.8: FASHION Benchmark Model



(a) Training Loss (b) Testing Accuracy

Figure 3.9: SVHN Benchmark Model

In Table 3.2 and Table 3.3, we provide further insights about results reported in Figure 3.7 by showing the number of samples per dataset before and after the data selection process. Table 3.2 summarizes results where SVHN is the target dataset and CIFAR, MNIST, and FASHION act as the noise datasets. One can observe that approximately 96% of samples belonging to SVHN are selected by our data selection approach. Another interesting observation is that approximately 60% of MNIST samples were also selected. This is because majority of MNIST images which are *handwritten-digits* are semantically close to *printed digits* (i.e., SVHN) and thus can be relevant to classify SVHN samples. This shows that our approach is also able to incorporate some of the "beneficial" noise. Finally, we can see that most of CIFAR and FASHION samples, which are not semantically close to SVHN's

images, were not selected by our approach. Table 3.3 reports results for experiments with FASHION as the target dataset and SVHN, CIFAR, and MNIST as the noise. In this case, where none of the datasets acting as noise are semantically close to FASHION's images, 84% of the samples belonging to the dataset considered as noisy are removed. Furthermore, 90% of FASHION samples are selected for training.

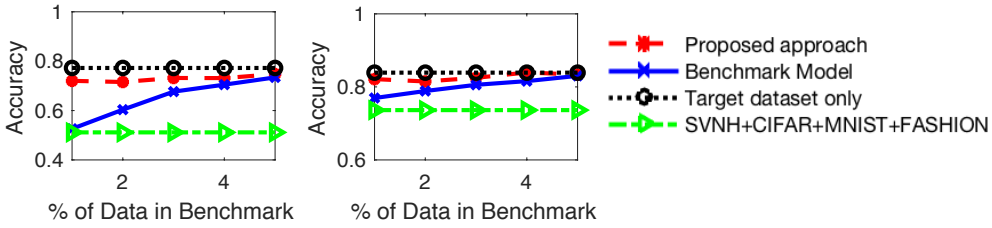
Table 3.2: Target SVHN

Dataset	# Samples Before Selection	# Samples After Selection
SVHN	73'257	70'336
CIFAR	50'000	8'236
MNIST	60'000	36'402
FASHION	60'000	19'036

Table 3.3: Target FASHION

Dataset	# Samples Before Selection	# Samples After Selection
SVHN	73'257	5'325
CIFAR	50'000	8'498
MNIST	60'000	8'807
FASHION	60'000	54'380

Additionally, Fig. 3.10 shows the results for varying sizes of the benchmark dataset. The benchmark models still suffer when the benchmark dataset is very small, whereas our data selection method performs well, indicating that our method enables effective federated learning even under strong noise levels and small benchmark dataset.



(a) Target: SVHN (b) Target: FASHION

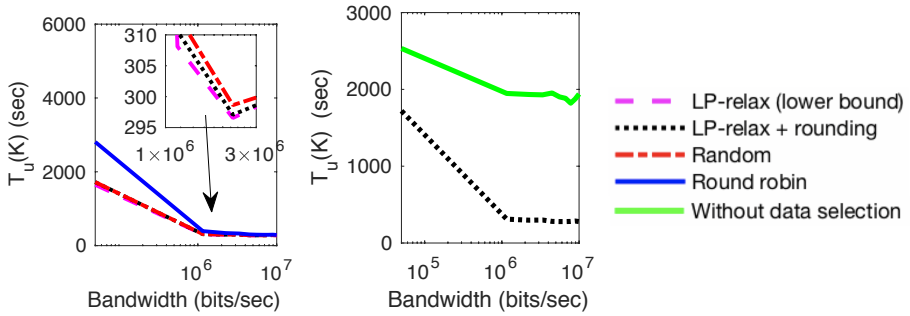
Figure 3.10: Varying size of benchmark data with strong noise.

3.6.2.3 Scheduling

We evaluate our scheduling algorithm in the following. We first consider the same settings as in Section 3.6.2.2, where each client has a mix of SVHN, FASHION, MNIST, and CIFAR-10. Futhermore, we assume four models M1, M2, M3, and M4, targeting SVHN, FASHION, MNIST, and CIFAR-10, respectively. Table 3.4 reports the real measurements of the computation time c_k on the Raspberry Pi Version 4 devices, where b_{kn} is the mini-batch size of task k at client n, which is different depending on k and n. The values of d_k and u_k are proportional with the model sizes by a factor of $\beta > 0$, and we assume that $d_k = \gamma u_k$ for $\gamma > 0$ (see Table 3.4). Note that $\frac{1}{\beta}$ can be considered as the uplink communication bandwidth in bits-per-second, and the downlink communication bandwidth is $\frac{1}{\gamma\beta}$.

Table 3.4: Values of d_k, c_k, and u_k for models M1-M4 obtained from real measurements.

	u_k (sec)	c_k (sec)	d_k (sec)
M1	$\beta \times 555,178 \times 32$	$b_{kn} \times 555,178 \times 2.84 \times 10^{-8}$	$\gamma \times u_k$
M2	$\beta \times 430,698 \times 32$	$b_{kn} \times 430,698 \times 2.84 \times 10^{-8}$	$\gamma \times u_k$
M3	$\beta \times 555,178 \times 32$	$b_{kn} \times 555,178 \times 2.84 \times 10^{-8}$	$\gamma \times u_k$
M4	$\beta \times 430,698 \times 32$	$b_{kn} \times 430,698 \times 2.84 \times 10^{-8}$	$\gamma \times u_k$



(a) (b)

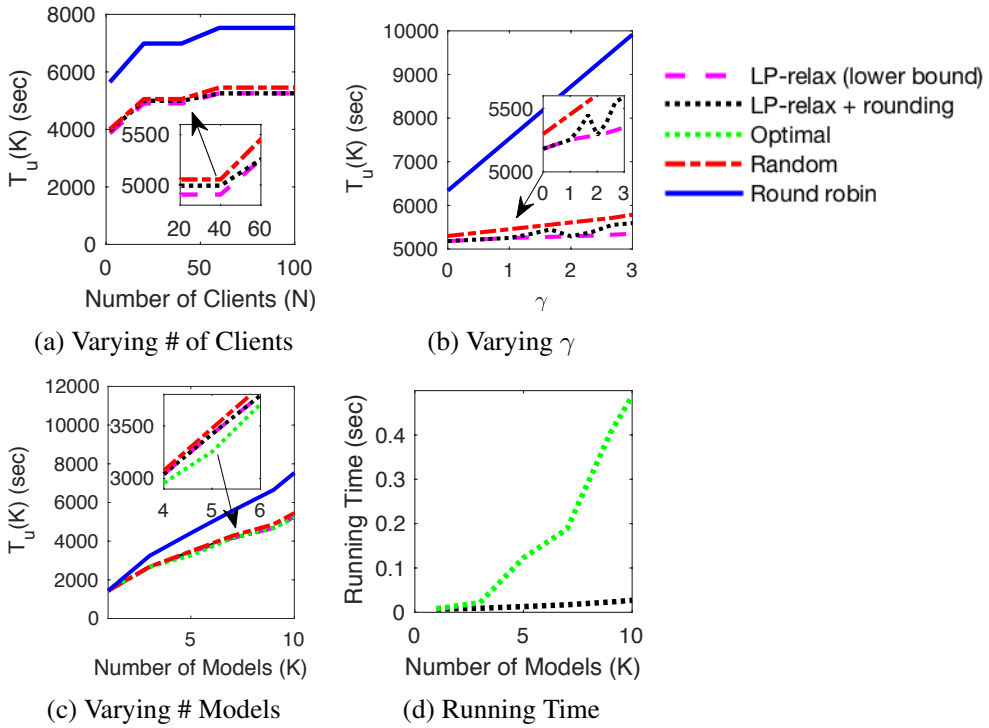
Figure 3.11: $T_u(K)$ under different bandwidth $\frac{1}{\beta}$, with $N = 100$, $K = 4$, and $\gamma = 1$, using real measurements in Table 3.4: (a) comparison of different scheduling methods with data selection, (b) comparison of LP relax + rounding with and without data selection.

Fig. 3.11 shows the completion times of one cycle (i.e., the maximum $T_u(K)$ among all N clients) under different bandwidth $\frac{1}{\beta}$, with $N = 100$, $K = 4$, and $\gamma = 1$, using real measurements in Table 3.4. We see in Fig. 3.11a that our proposed LP relax + rounding approach outperforms the random and round robin baselines. Although our approach outperforms the random scheduling approach by only a few seconds in each cycle, the aggregated saving of running multiple cycles in the federated learning process is still significant[6]. In Fig. 3.11b, we compare the completion time of each cycle with and without our data selection approach, when using our proposed LP relax + rounding scheduling algorithm. We see that data selection also reduces the training time, because by having only a subset of relevant data involved in each task, the mini-batch sizes at clients are smaller than involving all the data at clients, as we use a fixed percentage of the data size as the mini-batch size (see Section 3.3).

Next, we consider a simulated setting with larger varieties of configurations. For each client, we randomly generate a model size between 0 and $800,000$, a per-parameter processing speed between 3.2×10^{-7} and 3.2×10^{-9}, and a mini-batch size

[6]Note that more than 1000 cycles are often needed for training a good deep learning model. For instance, see Figs. 3.5 and 3.7 and note that each cycle includes $\tau = 10$ iterations, thus $20,000$ iterations includes $2,000$ cycles.

between 5 and 200. Similar to Table 3.4, the product of these three quantities is the simulated value of c_k, the value of u_k is the model size multiplied by 32 (number of bits per single-precision floating point number) then multiplied by β, and $d_k = \gamma u_k$. We set $N = 100$, $K = 10$, $\beta = 10^{-5}$, and $\gamma = 1$, unless stated otherwise. The results averaged over 10 different simulation runs are shown in Fig. 3.12. We see that our proposed approach performs better than the baselines with different number of clients, models, and different values of γ that captures the difference between the uplink and downlink bandwidths. Compared to solving the MILP in (3.11) optimally, which has an exponential time complexity, our approach significantly reduces the running time for obtaining the solution, as shown in Fig. 3.12d, while still providing a close-to-optimal result. Note that the running time in Fig. 3.12d is for solving the scheduling problem of a single client on Mac OS with 2.6 GHz Intel Core i7, 16 GB 2400 MHz DDR4.



(a) Varying # of Clients (b) Varying γ

(c) Varying # Models (d) Running Time

Figure 3.12: $T_u(K)$ in simulation environment.

3.7 Conclusion

In this chapter, we have considered a challenge in federated learning where each client may have various types of local data with noisy labels. To overcome this challenge, we have proposed a method for selecting the subset of relevant data to be involved in a federated learning task. In addition, we have formulated the scheduling of multiple federated learning tasks as an MILP, which is NP-hard, and proposed an efficient algorithm based on LP-relaxation and rounding to find its approximate solution. Through extensive experimental analysis using multiple real-world datasets, we demonstrate the effectiveness of the data selection method in strong open-set noise setting, as well as its advantages over multiple baseline approaches. The effectiveness and efficiency of our scheduling mechanism are further validated using experiments based on real system measurements as well as in a simulated setting.

Table 3.5: Summary of main notation for Chapter 3

Notation	Meaning
$\mathcal{D}_n$ / $\mathcal{D}$	Dataset at client n / Union of all datasets
(x_i, y_i)	Data sample i
N	Number of clients
θ_n / $\theta_{\mathcal{B}}$/ θ	Local parameter at client n/ Benchmark parameter/ Global parameter
$f(x_i, \theta)$	Model logic
$l(f(x_i, \theta), y_i)$	Individual local loss function
$L_n(\theta)$ / $L(\theta)$	Local loss function / Global loss function
$\mathcal{B}$	Benchmark dataset
V	Reference distribution of loss values
P_n / P	Loss values set clients n / Union of all clients
$F_P(x)$	Cumulative distribution function of P
$F_V(x)$	Cumulative distribution function of V
$G(.,.)$	Kolmogorov-Simrnov Distance
λ	Threshold in loss values
$\mathcal{F}_n$ / $\mathcal{F}$	Set of relevant samples at node n / Union of relevant samples
$z_{k,j}$	Binary variable equal to 1 is model k is scheduled in slot j
d_k	Download duration of model k
c_k	Computation duration of model k
u_k	Upload duration of model k
c_k	Computation duration of model k
$T_d(j)$	Duration of the cycle starting from the beginning until downloading in the j-th slot
$T_c(j)$	Duration of the cycle starting from the beginning until computing in the j-th slot
$T_u(j)$	Duration of the cycle starting from the beginning until uploading in the j-th slot

CHAPTER 4

Efficient Continual Learning Using Bayesian Approaches

4.1 Introduction

In federated learning systems, local training usually happens when clients/devices have adequate bandwidth and energy (e.g., battery power), More specifically, training happens only when *the device is idle, charging, and connected to an unmetered network such as Wi-Fi* [46]. As a result, it is not uncommon for some active devices to drop out at some point before the completion of the learning task due to connectivity or energy constraints. Clients leaving the learning collaboration can have a major impact on the global model. If the data on the leaving clients is very different from others, the knowledge from this "missing" clients will be forgotten over time due to the *catastrophic forgetting* phenomenon (i.e., the global model will be updated without taking into account parameters from missing clients). In other words, as devices drop out or join, data available to train the global model varies over time (i.e., non-stationary), which causes the global model to forget previously acquired knowledge when learning new information.

The field of continual learning (also referred to as incremental learning or lifelong learning) addresses the catastrophic forgetting problem by enabling models to acquire new knowledge from continuous input while preventing the latest input from significantly interfering with existing knowledge. In recent years, many approaches

have been proposed to trade-off between adapting to the most recent data while at the same time retaining old knowledge.

However, most existing approaches require to be informed when data distribution changes during training (often referred to as *task shifts*) to take some actions. Assuming knowledge of task boundaries is a strong assumption as this kind of setup is rarely encountered in practical scenarios where task shift boundaries are almost always poorly defined, and task identities are often unknown. In the federated learning setting, knowing the task shift would mean being able to predict exactly when a client will drop out or join the system, which is also very impractical. Among the few existing techniques for continual learning without assuming known task boundaries (often referred to as *task free* setting), they all require storing raw training samples that violate the main property of federated learning and consequently data privacy policies. This emphasizes the need for continual learning techniques to support continuous federated learning where clients may continuously join and leave the collaboration over time.

This chapter introduces a task-free continual learning approach that does not require storing training data, making it applicable to federated learning. It is important to highlight that we define a new task as a shift in the input data distribution while the outputs (i.e., classes/labels) remain the same among the different tasks. We do not consider class-incremental learning scenarios (i.e., where new classes are added for each new task).

Our method uses Bayesian neural networks as the classifier model as they have favorable properties for incremental learning. A new classifier is trained each time when a new task is detected. Classifiers are saved in a buffer of fixed storage capacity for the purpose of maintaining good accuracy over time. At testing time, predictions from the stored classifiers are merged. Our main contributions in this chapter are as follows:

- A method based on Bayesian Neural Networks (BNNs) to continually learn on new data while minimally forgetting what has been learned previously,
- A mechanism to automatically detect shifts in data, which allows the algorithm to work without assuming known task boundaries
- An efficient mechanism for keeping the model within a maximum size, so that we do not exceed the storage capacity, and
- Validation of our approach on different continual, learning scenarios with real datasets.

The remaining of the chapter is organized as follows. In Section 4.2, we review the related work. In Section 4.3, we introduce some preliminary definitions. Our proposed approach is described in Section 4.4. Our experimental results are reported in Section 4.5. Finally, in the last section, we draw our conclusion. The summary of notations used in this chapter is presented in Table 4.9.

4.2 Related Work

Approaches for continual learning mainly use three strategies:

- *Expansion* strategies accommodate the new knowledge by growing the capacity of the model. Authors in [47] develop an approach, referred to as *progressive network* that blocks/fixes the network trained on previous data, and expands the architecture to allocate the new information.
- *Rehearsal/Replay* strategies [48, 49, 50, 51, 52, 53] store a few representative training data points over time and replay them when training the model with new data. These strategies have privacy issues as they require to store raw data [54].

- *Regularization* strategies [55, 56, 57, 52, 58, 48] have the main idea of adding a term (often called a regularization term) to the loss function, to prevent the model from varying significantly when learning on new data.

Much of existing work uses a hybrid combination of different strategies. For example, a combination of rehearsal and regularization techniques is used in [48, 52]. Alternatively, one can also distinguish the two different settings for continual learning:

- *Task-based* setting, where there is a sequence of tasks with known boundaries and identities, and at each time one task is learned; and
- *Task-free* setting where task shifts are poorly defined (i.e., the task shifts may not happen abruptly but rather gradually) and task identities are unknown.

Most of existing approaches [57, 49, 52, 47, 48, 53, 55, 56, 59, 60, 50, 51, 58] are *task-based* approaches as they require to be informed when data distribution is switched during continual learning, so that the algorithm can act accordingly. Assuming knowledge of tasks boundaries is a strong assumption as this kind of setup is rarely encountered in practical scenarios.

Despite its importance, the *task-free* setting has been largely understudied, except for a few pieces of work that we describe next. Most of the existing task-free approaches are based on replay/rehearsal strategies [61, 62, 63, 64], where a small buffer of data samples, often referred to as episodic memory, is used for the rehearsal purpose. Hybrid replay and expansion methods also exist [65], where expansion alone is not enough to avoid catastrophic forgetting and replay is required. [66] introduced a task-free expansion method governed by Bayesian non-parametric approaches to automatically determine when to expand, which has shown to outperform many other task-free approaches, such as those in [62, 64]. However, the disadvantage of this approach is that it is very sensitive to the choice of hyper-parameters.

Furthermore, it also requires a short-term memory that collects some training data during the process. Unlike these approaches that employ either a short-term memory or replay buffer to store training samples, the approach we propose here does not require storing any data point, which is a strong advantage as storing raw training samples causes privacy issues as emphasized by [54].

Our work also shares some similarities with the work by [60], which also uses BNNs to perform continual learning by adapting the learning rate based on the uncertainty of network parameters. However, this approach requires each task to be trained until reaching a "plateau". The learning rate adaptation may also slow down the training process. Finally, most experiments reported in their work require task identity knowledge during inference. We shall show that our relatively simple approach, which does not require task identity knowledge for either training or inference, is able to outperform existing task-free approaches in performance, storage, and speed.

4.3 Preliminaries

4.3.1 Task-Free Continual Learning

We assume that there are K tasks. Each task corresponds to a specific distribution of training and testing data, where the data distributions for different tasks are generally different. At any point in time, only one mini-batch of data from the dataset of an arbitrary task $k \in \{1, 2, ..., K\}$ is revealed and available to the system. We consider the task-free setting where we do not know from which task each mini-batch of data comes from, but the algorithm that we present later estimates the task boundary and identity. Our system does not save any data point other than the current mini-batch of data.

The K tasks arrive in a sequential manner and each task may appear one or

multiple times during the training phase. The goal of training is to learn a model that can accurately classify data from any of the K tasks. During the inference phase, the system receives mini-batches of the testing data from arbitrary tasks. The task identity is unknown to the system in both training and inference phases.

4.3.2 Bayesian Neural Networks (BNNs)

BNNs have useful characteristics that are beneficial for continual learning. Unlike standard DNNs that aim to find a deterministic value of model parameters to fit the data, BNNs aim to estimate the posterior distribution $p(\mathbf{w}|\mathcal{D})$ over the model parameter vector $\mathbf{w}$, by considering the training data $\mathcal{D}$, which includes the input data and the target output for a supervised classifier, as the evidence. This posterior distribution can be estimated using Bayes' rule as $p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$, where $p(\mathbf{w})$ is an assumed prior distribution and $p(\mathcal{D}|\mathbf{w})$ is the likelihood.

In most cases, the true posterior $p(\mathbf{w}|\mathcal{D})$ is intractable because the term $p(\mathcal{D})$ cannot be efficiently computed in practice. Variational methods are used to find a parametric distribution $q(\mathbf{w};\phi)$ that approximates $p(\mathbf{w}|\mathcal{D})$, i.e., $p(\mathbf{w}|\mathcal{D}) \approx q(\mathbf{w};\phi)$. The parametric distribution q belongs to a "well-behaved" family of distributions, such as normal or exponential distributions that can be represented by a set of parameters ϕ such as mean μ and variance σ^2 (i.e., $\phi = (\mu, \sigma)$). The goal of BNN training is to find the parameter ϕ that minimizes the Kullback-Leibler (KL) divergence between the parametric distribution q and the true posterior distribution: $\phi^* = \arg\min_\phi KL(q(\mathbf{w};\phi)||p(\mathbf{w}|\mathcal{D}))$. The KL divergence is often intractable. However, minimizing the KL divergence is equivalent to minimizing the negative of the tractable evidential lower bound (ELBO) [67]:

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\phi)}[\log q(\mathbf{w};\phi) - \log p(\mathbf{w}) - \log p(\mathcal{D}|\mathbf{w})]. \tag{4.1}$$

At the end of training, $q(\mathbf{w};\phi)$ can be used as an approximation of $p(\mathbf{w}|\mathcal{D})$. In practice, $\mathcal{L}(\phi,\mathcal{D})$ is often minimized using stochastic gradient descent (SGD) on ϕ where each iteration is based on an approximate gradient of $\mathcal{L}(\phi,\mathcal{D})$ computed on a mini-batch sampled from $\mathcal{D}$.

4.3.3 Continual Learning with BNNs.

In theory, the Bayesian framework has an interesting property that when learning from sequential tasks with training data $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K$, one can estimate the posterior distribution $p(\mathbf{w}|\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k)$ for $k \in \{2, ..., K\}$ by using the previous posterior as the new prior:

$$p(\mathbf{w}|\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k) \propto p(\mathcal{D}_k|\mathbf{w})p(\mathbf{w}|\mathcal{D}_1, ..., \mathcal{D}_{k-1}). \tag{4.2}$$

Unfortunately, as mentioned earlier, the true posterior is intractable in most practical scenarios and performing repeated approximations accumulates errors, which causes the algorithm to forget old tasks. This underlines the need for new approaches for efficient and effective continual learning.

4.4 Our Approach

The main idea of our approach is that we train multiple BNN models, each of which captures one or multiple similar tasks. These models can be regarded as experts for such tasks. We save up to N representative models during training, which are then used for inference, where N is a user-defined positive integer related to the desired aggregated model size and complexity. We emphasize that our system *does not* know true task boundaries or identities during either training or inference.

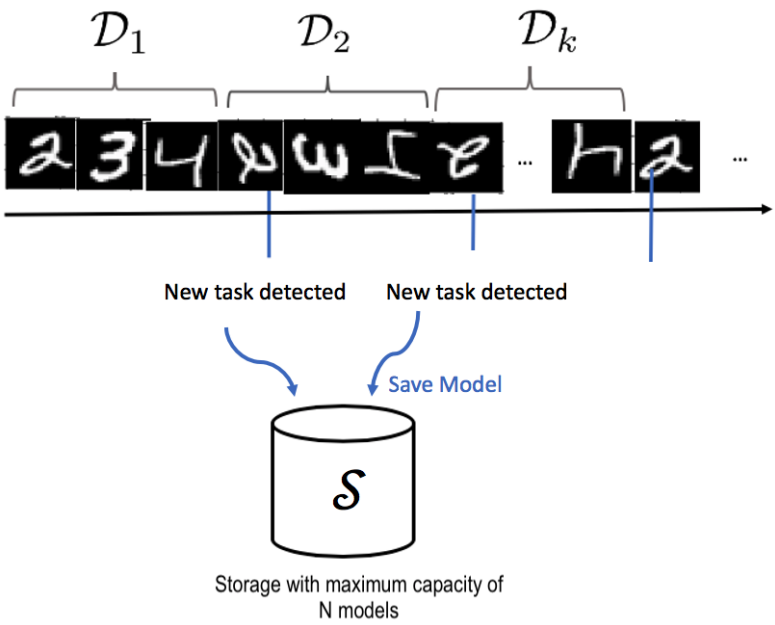


Figure 4.1: Overview of training procedure.

4.4.1 Training

The training procedure of our approach has three main components: detecting task boundaries (Section 4.4.1.1), knowledge transfer between tasks (Section 4.4.1.2) and model management (Section 4.4.1.3). In the following, we describe these components in detail.

4.4.1.1 Detecting Task Boundaries

As shown in Figure 4.1, during training, mini-batches of training data arrive in a sequential manner, where we do not know from which task the data comes from. Upon receiving a new mini-batch, the variational parameters ϕ is updated after each SGD iteration on a minibatch (4.1). The task boundaries are detected based on the changes of values of the log-likelihoods across mini-batches. In other words, we estimate that a new task has started if the average log-likelihood on the most recent mini-batches is sufficiently smaller than that on some previous mini-batches. To do so, we consider the expected log-likelihood term in (4.1), $l_t := \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\phi_{t-1})}[\log p(\mathcal{D}_{(t)}|\mathbf{w})]$ with $\mathcal{D}_{(t)}$

being the current mini-batch of data, which can be approximated using Monte Carlo by sampling $\mathbf{w}$ from $q(\mathbf{w}; \phi_{t-1})$. We track the expected log-likelihood term l_t over time and if it has decreased by a significant margin we detect a new task.

More specifically, we define a time window of length T and in any iteration $t = nT$ ($n \in \mathbb{Z}^+$) we compute $L_n := \frac{1}{T} \sum_{\tau=(n-1)T+1}^{nT} l_\tau$. Then, we compare L_n with L_{n-1}. If $L_{n-1} - L_n > \theta$ at iteration $t = nT$, where θ is some threshold, we say that the likelihood has decreased by a significant margin and a new task has started at $t = (n-1)T + 1$ (and the previous task ends at $t = (n-1)T$). The reason for computing L_n on a time window T is to average out the noise that can exist in a single mini-batch.

Our empirical finding suggests that a proper choice of θ is the standard deviation of L_n over a few recent time windows starting at or after the start of the current task.

4.4.1.2 Knowledge Transfer Between Tasks

When a task switch is detected, we can use the posterior of $\mathbf{w}$ learned from the previous task as the prior of the current task, as in (4.2), so that we transfer knowledge from previous tasks into the current model to some degree. Such knowledge transfer is optional and depends on whether there may be correlation across different tasks. Our empirical results show that knowledge transfer may benefit certain scenarios while not in other scenarios (See Section 4.5).

4.4.1.3 Model Management

When a task ends at the t_k-th iteration, we obtain ϕ_{t_k} that captures the variational parameters for the model obtained for task k. We save up to N of such models that are jointly used for inference as we will see in Section 4.4.2. In addition, at any iteration $t = nT$, we save the model ϕ_{nT} in a buffer if a new task is not detected, so that we can revert back to this model later if a new task is detected after the next

window $t = (n+1)T$.

Let $\mathcal{S}$ denote an *indexed set* of stored models and we always ensure $|\mathcal{S}| \leq N$. We use the task index k to denote the model ϕ_{t_k} for simplicity. When a new task $k+1$ is detected, if $|\mathcal{S}| < N$, we always save the model for the previous task k into $\mathcal{S}$. When $|\mathcal{S}| = N$, i.e., we have reached the storage (aggregated model size) limit N, we determine which models to keep in $\mathcal{S}$ in the following way.

Every time when a new task $k+1$ is detected, we compute a distance (e.g., KL divergence) between the softmax outputs[1] of the BNN model k and all models in $\mathcal{S}$, evaluated on the current mini-batch. Let k_i denote the i-th model currently stored in $\mathcal{S}$, i.e., $k_i \in \mathcal{S}$, where we note that we may have $i < k_i$ because some models before k_i may have been deleted. To avoid confusion, we write the current task (and its corresponding model) as $k_\bullet := k$. Let d_{k_i,k_j} denote the distance between models k_i and k_j computed on the current mini-batch. A small (correspondingly, large) value of d_{k_i,k_j} means that models k_i and k_j give similar (correspondingly, different) predictions on current mini-batch. By computing $d_{k_i,k_\bullet}$ for all $i \in \mathcal{S}$ every time when a new task $k_\bullet + 1$ starts, we can progressively obtain all the distances d_{k_i,k_j} for all $i \in \{1, ..., |\mathcal{S}|\}$, $j \in \{1, ..., |\mathcal{S}|, \bullet\}$, and $i < j$ (we assume $\bullet > |\mathcal{S}|$). Hence, at any point in time, the system keeps the following $|\mathcal{S}|$-by-$(|\mathcal{S}|+1)$ distance matrix:

$$\mathbf{A} = \begin{bmatrix} d_{k_0,k_1} & d_{k_0,k_2} & \cdots & d_{k_0,k_{|\mathcal{S}|}} & d_{k_0,k_\bullet} \\ \infty & d_{k_1,k_2} & \cdots & d_{k_1,k_{|\mathcal{S}|}} & d_{k_1,k_\bullet} \\ \infty & \infty & \ddots & & \vdots \\ \infty & \infty & \infty & d_{k_{|\mathcal{S}|-1},k_{|\mathcal{S}|}}) & d_{k_{|\mathcal{S}|-1},k_\bullet} \\ \infty & \infty & \infty & \infty & d_{k_{|\mathcal{S}|},k_\bullet} \end{bmatrix} \tag{4.3}$$

which is an upper-triangle matrix because we cannot compute newer models on older mini-batches as we do not save data. Note that each column of A is evaluated on

[1] Similar to most of existing work, we consider supervised classification problems in this work.

different a mini-batch, however we omit mini-batch indices in order to simplify the notation. For convenience of minimization, we set those distances that cannot be computed as infinity. Note that if $k_i \in \mathcal{S}$ at the end of $k_\bullet$, we must have $k_i \in \mathcal{S}$ at the end of k_j for any $j > i$. Hence, the above matrix $\mathbf{A}$ can be computed progressively every time when a new task is detected, without saving any data.

When $|\mathcal{S}| = N$, we have $N + 1$ models including $k_\bullet$, and we need to delete a model. To determine which model to delete, we find the smallest distance d_{k_i,k_j} in $\mathbf{A}$ and its corresponding pair of models k_i and k_j. These two models are the most similar to each other. We delete the older model k_i ($i < j$) because newer models can capture some information of older models when knowledge transfer is enabled. After deleting k_i, the distances related to k_i are also deleted. Here, we note that when a model k_m is deleted from $\mathcal{S}$, it will never be added back again and we decrement the indices i and j by one for all $i, j > m$. With this process, we never delete $k_\bullet$ because it is the newest, so we add $k_\bullet$ to $\mathcal{S}$.

4.4.2 Inference

At inference time, for each stored model $k \in \mathcal{S}$, we use Monte Carlo approximation to estimate the probability distribution of the output (label) $\mathbf{y}$ for a given input data sample $\mathbf{x}$:

$$p_k(\mathbf{y}|\mathbf{x}) = \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\mathcal{D}_k)} p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \approx \frac{1}{R} \sum_{r=1}^{R} p(\mathbf{y}|\mathbf{x}, \mathbf{w}_k^{(r)}) \tag{4.4}$$

where $\mathbf{w}_k^{(r)}$ is sampled from $q(\mathbf{w}; \phi_{t_k}^*)$ and we take the average over R samples. By computing the above for all models in $\mathcal{S}$, we obtain the approximate probability distribution $p_k(\mathbf{y}|\mathbf{x})$ from each model $k \in \mathcal{S}$. Assume that the ground-truth label $\mathbf{y}$ includes one-hot encoded labels, our final goal is to find the predicted label $\hat{\mathbf{y}}$ that is equal to the ground-truth label $\mathbf{y}$ with high probability.

Based on the estimated probability distributions $\{p_k(\mathbf{y}|\mathbf{x}) : \forall k \in \mathcal{S}\}$, we find $\hat{\mathbf{y}}$ in the following way. We first compute an uncertainty value defined as $u_k = \sum_d \sigma_k([p(\mathbf{y}|\mathbf{x}, \mathbf{w})]_d)$, where $[p(\mathbf{y}|\mathbf{x}, \mathbf{w})]_d$ is the d-th element of the probability distribution $p(\mathbf{y}|\mathbf{x}), \mathbf{w})$ (i.e., probability assigned to the d-th class/label) and $\sigma_k(\cdot)$ denotes the standard deviation (estimated over R samples) when sampling $\mathbf{w}$ from $q(\mathbf{w}; \phi^*_{t_k}) \approx p(\mathbf{w}|\mathcal{D}_k)$. Then, we find the model k^* with the smallest uncertainty, i.e., $k^* := \arg\min_k u_k$; and afterwards find a one-hot encoded $\hat{\mathbf{y}}$ that has the highest probability, i.e., $\hat{\mathbf{y}} := \arg\max_{\mathbf{y}:\|\mathbf{y}\|_0=1} p_{k^*}(\mathbf{y}|\mathbf{x})$.

The rationale behind this approach is that, as the uncertainty u_k is defined as the sum standard deviation of each element y_d of $\mathbf{y}$, it captures how certain each model k is about its prediction. The most certain model most likely aligns well with the task that generats the input data.

4.4.3 Application to federated learning

The continual learning technique proposed above can be applied to general (centralized) machine learning systems. Furthermore, as our continual approach does not require to save training data samples it can be easily extend to federated learning systems. In federated learning settings, our approach works as follows: each client trains on its local dataset in a similar fashion as described in section 4.4 and manages the storage of its own buffer where a fixed number of local BNN model can be stored. At synchronization time, each client sends its set of stored BNN models to the central server. The server is able to make predictions by merging model predictions from the different models received from each clients, in a similar fashion as described in Section 4.4.2. When a client leaves the collaboration, the global model (i.e. central server) can still uses the last received BNNs in order not to forget its contribution. Even if the BNNs are not be up to date, using them will still ensure that what was learned before is not forgotten.

4.5 Experiments

4.5.1 Continual Learning Scenarios

We experiment our proposed approach in various continual learning scenarios common in the literature. We consider non-class-incremental learning scenarios where new task are defined as a shift in the input data distribution while the outputs (i.e., classes/labels) remain the same among the different tasks. The considered scenarios are:

- **Rotated MNIST** [49], where each task is represented by digits rotated by a fixed angle. In most of our experiments, we define 4 tasks with angles $0°$, $30°$, $60°$, and $90°$. , which is more challenging than the scenario used by [68] with 5 tasks and smaller rotation angles $0°$, $10°$, $20°$, $30°$, and $40°$ (results for this alternative setting are also reported in Table 4.4). Each task has 60,000 training data samples.

- **Permuted MNIST** [69], where each task is a certain random permutation of the input pixels of MNIST images. The distribution of labels remains the same but the distribution of input images is different. Similar as [60], we learn a sequence of 10 random permutations. Each task has 60,000 training data samples.

- **MNIST-SVHN** [53] consists of two tasks, one task with MNIST data and the other task with SVHN data [41]. The two tasks have 60,000 and 73,000 training data samples, respectively.

4.5.2 Compared Methods

For our proposed approach, we report two sets of results: with knowledge transfer (**Ours-T**) and with no knowledge transfer (**Ours-NT**). We compare with several

other state-of-the-art *task-free* approaches:

- **Reservoir** [64], which is a simple experience replay method that has been shown to outperform many other continual learning approaches. The episodic memory can be managed in a task-free setting, which makes Reservoir a strong baseline to compare with, in terms of both accuracy and complexity.

- **CN-DPM** [66], which is an expansion-based method that consists of a set of experts that learn different subsets of the data that belong to different tasks. The number of experts is governed by a non-parametric Bayesian framework. We also include an extension of CN-DPM that leverages task-homogeneity within a mini-batch (i.e., data in the same mini-batch come from the same task, see Section 4.4.2) during inference to select the best expert, which is referred to as CN-DPM-H.

- **Fine-Tuning**, which is a popular baseline used in previous works where the model is naively trained using SGD or its accelerated variants, without paying specific attention to avoiding catastrophic forgetting. The model is an ordinary neural network.

4.5.3 BNN Architecture and Other Details

For our approach, we use a BNN with a single fully-connected layer containing 64 neurons. We use Adam optimizer with a mini-batch size of 64 and learning rate of 0.001 for all the experiments. Apart from the experiment related to model management, the capacity (i.e., N) is equal to the number of tasks. At inference time, the number of Monte Carlo samples is set to either $R = 10$ (referred to as "Ours-*-10") and the inference batch size is set to 1 unless state otherwise. Our BNN implementation is based on the code by [70][2]. For our method, we initialize the prior as a normal

[2] https://github.com/kumar-shridhar/PyTorch-BayesianCNN

distribution $\mathcal{N}(0,1)$ for Rotated-MNIST, and Permuted-MNIST, and $\mathcal{N}(0,0.01)$ for MNIST-SVHN. The variational distribution q is also a normal distribution $\mathcal{N}(\mu,\sigma^2)$, where the initial mean μ is randomly sampled from $\mathcal{N}(0,0.01)$ and the initial standard deviation is chosen such that $\sigma = \log(1+e^{\rho})$ where ρ is sampled $\mathcal{N}(-5,0.1)$, as suggested in the code by [70]. The time window length T for task detection is set to the number of mini-batches in one epoch[3]. The results of Permuted MNIST with CN-DPM was obtained by setting the learning rates of both the generator and discriminator to $1/4$ of the original learning rates used for the MNIST dataset in the original paper by [66]. The reason is that, without this change, the training process with CN-DPM gets into ill-conditions that cause large fluctuations in the loss, ultimately generating a very large number of experts making the training excessively slow.

For baseline approaches, we replicate the same neural network architectures and other settings as [66]. All baselines results were reproduced using the original code[4] from the paper by [66].The system receives data from each task for a consecutive duration of 10 epochs during training as in [66], before switching to the next task.

4.5.4 Performance Evaluation

After training on tasks arriving sequentially, the model is evaluated on the test data from the union of all tasks. We compute the task-wise accuracy a_k, which is the accuracy of task k's test data after having sequentially learned all K tasks, and the overall accuracy $\overline{a} := \frac{1}{K}\sum_{k=1}^{K} a_k$. We also record the number of model parameters (floating point numbers) and the number of stored training data samples for each method. For our approach, the number of parameters is the total number of BNN

[3] An epoch denote the entire processing by the learning algorithm of the entire train-set. The number of mini-batch required to complete an epoch is the size of the training dataset divided by the size of the mini-batch.

[4] https://github.com/soochan-lee/CN-DPM

parameters (mean and standard deviation values) of *all* models stored in $\mathcal{S}$. We further record the per-sample time of training and inference involving all K tasks, measured on a cloud computing instance with 8 CPU cores, 8 GB memory, and a K80 GPU.

We obtain statistics from 5 independent runs with different random seeds for each setting, and report mean and standard deviation values.

4.5.5 Results

The results on rotated, permuted MNIST, and MNIST-SVHN are shown in Tables 4.1 to 4.3, with comparison to the Fine-Tuning, Reservoir, and CN-DPM techniques. We see that our method generally outperforms the baselines in terms of accuracy and the number of model parameters. Furthermore, our approach does not require storing any training data samples, unlike Reservoir and CN-DPM. This is an advantage in terms of not only privacy but also storage. To illustrate, for the MNIST rotate dataset, saving 500 samples for Reservoir corresponds to storing 3M floating point numbers (i.e., $28 \times 28 \times 1 \times 500 \times 10$). The poor performance of CN-DPM is possibly related to hyper-parameter tuning. Although we use the same hyper-parameters as in the original paper for MNIST dataset, it does not seem to perform well for rotated and permuted MNIST. As the approach is very slow (especially since each task has 60,000 training samples), it is impractical to try a wide range of hyper-parameters. This is a shortcoming of CN-DPM requiring detailed hyper-parameter tuning, which is often impractical.

Surprisingly, our approach without knowledge transfer (i.e., Ours-NT-10) outperforms our approach with knowledge transfer (i.e., Ours-T-10) for all experiments. This might be due to the fact that in practice the true posterior is intractable which causes repeated approximation of the posterior to accumulate errors. Consequently, setting the prior of the new task to the posterior of the previous tasks seems to add

more noise than when just setting the prior of the new detected task as a normal distribution.

Table 4.1: Rotated MNIST

Method	Overall Acc. (%)	#Param.	#Samples	Training time per sample (ms)	Inference time per sample (ms)
Fine-tuning	61.93±0.73	478K	0	0.454±0.007	0.272±0.002
Reservoir	88.49±0.79	478K	500×4	0.507±0.007	0.254±0.004
CN-DPM	53.15±0.92	709K	500	3.984±0.149	0.373±0.004
Ours-NT-10	89.76±0.43	407K	0	0.190±0.004	2.051±0.097
Ours-T-10	89.38±0.64	407K	0	0.153±0.006	1.722±0.036

Table 4.2: Permuted MNIST

Method	Overall Acc. (%)	#Param.	#Samples	Training time per sample (ms)	Inference time per sample (ms)
Fine-tuning	42.62±1.54	478K	0	0.448±0.003	0.189±0.006
Reservoir	87.47±0.60	478K	500×10	0.571±0.032	0.216±0005
CN-DPM	14.07±3.51	1.19M	500	6.115±0.954	0.408±0.077
Ours-NT-10	89.04	1.02M	0	0.166±0.002	0.762±0.010
Ours-T-10	83.78±0.35	1.02M	0	0.156±0002	0.69±0.017

Table 4.3: MNIST-SVHN

Method	Overall Acc. (%)	#Param.	#Samples	Training time per sample (ms)	Inference time per sample (ms)
Fine-tuning	85.78±2.83	11.2M	0	6.49±0.031	0.798±0.003
Reservoir	94.90±0.14	11.2M	1000×2	21.510±0.841	0.808±0.015
CN-DPM	95.38±0.12	7.8M	1000	10.179±0.073	2.776±0.018
Ours-NT-10	90.76±0.39	248K	0	0.342±0.012	2.214±0.033
Ours-T-10	89.82±1.03	248K	0	0.417±0.014	2.606±0.064

In Table 4.4, we compare our approach with some existing *task-based* approaches that require knowing the task identities during training, such as **Orthogonal Gra-**

Table 4.4: Comparison with task-based methods

Rotated MNIST					
Method	0°	10°	20°	30°	40°
OGD	75.6±2.1	86.6 ±1.3	91.7 ±1.1	94.3± 0.8	93.4±1.1
A-GEM	72.6±1.8	84.4 ±1.6	91.70 ±1.1	93.9± 0.6	94.6±1.1
EWC	61.9±2.0	78.1 ±1.8	89.0 ±1.6	94.4± 0.7	93.9±0.6
Fine-Tuning (SGD)	62.9±1.0	78.5 ±1.5	88.6 ±1.4	95.1± 0.5	94.1±1.1
Ours-NT	91.40±0.09	95.05±0.42	94.27±0.07	91.140±0.26	93.24±0.04
Ours-T	88.35±0.39	94.72±0.17	96.88±0.35	96.91±0.28	95.48±0.18

Permutated MNIST					
Method	T1	T2	T3	T4	T5
OGD	79.5±2.3	88.9 ±0.7	89.6±0.3	91.8± 0.9	92.4±1.1
A-GEM	85.5±1.7	87.0 ±1.5	89.6 ±1.1	91.2± 0.8	93.9±1.0
EWC	64.5±2.9	77.1 ±2.3	80.4 ±2.1	87.9±1.3	93.0±0.5
Fine-Tuning (SGD)	60.6±4.3	77.6 ±1.4	79.9 ±2.1	87.7±2.9	92.4±1.1
Ours-NT	95.13±0.18	94.63±0.02	93.36±0.03	92.37±0.13	93.68±0.05
Ours-T	95.71±0.20	97.07±0.11	96.22±0.07	83.99±0.37	88.49±2.90

dient Descent [68], an approach that restricts the direction of the gradient updates to avoid catastrophic forgetting, **A-GEM** that works by ensuring that the episodic memory loss over the previous tasks does not increase [71], **EWC** [57] which is a popular regularization technique using Fisher information to find the importance of weights, and **Fine-Tuning SGD** which is a similar baseline as Fine-Tuning used in the main paper, which shows what happens if nothing is done to avoid catastrophic forgetting. For these baseline approaches, we report the original results from the paper by [68] and run our approach for the same number of epochs (i.e., 5) as for the baselines in order to compare fairly. Table 4.4 shows that our *task-free* approach is also able to outperform the *task-based* baseline approaches. We emphasize here that even in this experiment, our approach is run without knowledge of task identities, for both training and inference, whereas all baselines have such knowledge during training.

In Table 4.5, we consider the case where the four different angles of rotated MNIST appear cyclically so that there are 12 tasks in total, i.e., the sequence of tasks are $0°, 30°, 60°, 90°, 0°, 30°, 60°, ...$ The results are obtained using our method with

Table 4.5: Varying capacity with cyclic Rotated MNIST (12 Tasks), per task and overall accuracies (in %)

Capacity (i.e., N)	0°	30°	60°	90°	Overall
2	71.55±0.71	92.44±0.09	89.55±0.98	94.01±0.14	86.69±0.14
4	86.43±0.08	92.14±0.10	93.93±0.12	91.32±0.24	90.95±0.07
8	86.56±0.10	92.17±0.21	93.97±0.23	91.35±0.07	91.01± 0.04
12	86.88±0.07	92.19±0.01	93.87±0.01	91.32±0.07	91.04±0.05

knowledge transfer (i.e., Ours-T). We consider the impact of storage capacity N. We see that choosing $N = 4$ gives close to the highest accuracy, since there are only four unique rotation degrees out of the 12 tasks. When we set $N = 2$, the overall accuracy is still good. Furthermore, we observe that our algorithm saves models for degrees $30°$ and $90°$ when $N = 2$, which reveals that our model management approach is effective because the $30°$ and $90°$ models may be still good for $0°$ and $60°$, respectively.

Tables 4.6 to 4.8 present results for different mini-batch sizes at inference time, with $R = 10$. Results shows that availability of task-homogeneous mini-batch for inference can improve our approach. One can observe that the accuracy can be improved when the size of inference batch size increases for all experiments.

Table 4.6: Rotated Mnist, Varying Mini-batch Inference Size

Method	Size: 1	Size: 2	Size: 4	Size: 8	Size: 16	Size: 32
Ours-NT-10	89.76±0.43	93.15±0.23	95.53±0.09	96.54±0.08	96.80±0.06	96.87±0.06
Ours-T-10	89.38±0.64	92.81±0.30	95.01±0.16	96.22±0.08	96.80±0.06	96.87±0.06

Table 4.7: Permuted Mnist, Varying Mini-batch Inference Size

Method	Size: 1	Size: 2	Size: 4	Size: 8	Size: 16	Size: 32
Ours-NT-10	89.04	94.13	95.52	95.60	95.64	95.65
Ours-T-10	83.78	88.68	90.96	91.45	91.53	91.53

Table 4.8: MNIST-SVHN, Varying Mini-batch Inference Size

Method	Size: 1	Size: 2	Size: 4	Size: 8	Size: 16	Size: 32
Ours-NT-10	90.76±0.39	92.17±0.22	93.18±0.15	93.60±0.19	93.64±0.17	93.66±0.18
Ours-T-10	89.82±1.03	91.61±0.56	92.91±0.31	93.49±0.16	93.60±0.17	93.61±0.16

4.6 Conclusion

In this work, we have proposed and studied a novel continual learning technique with three characteristics: 1) low complexity and fast running time, 2) no requirement of task identity knowledge in either training or inference (i.e., task-free), 3) no storing of training data samples. Our results have shown that our proposed method outperforms various state-of-the-art baselines in terms of accuracy, storage, and speed, for a variety of continual learning settings. Since our approach works in task-free settings and does not require storing training data samples, it can be readily applied to solve catastrophic forgetting problem (which is caused by clients dropping out during training) in federated learning.

Table 4.9: Summary of main notation for Chapter 4

Notation	Meaning
K	Number of tasks
$p(\mathbf{w})$	Prior distribution
$\mathcal{D}$	Data
$p(\mathbf{w}\|\mathcal{D})$	Posterior distribution
$p(\mathcal{D}\|\mathbf{w})$	Approximation of the posterior distribution
$q(\mathbf{w};\phi)$	Approximation of the posterior distribution
ϕ	Variational parameter
t	Iteration/mini-batch index
$\mathcal{D}$	Data
l_t	Expected log-likelihood term
N	Maximum number of models that can be stored
$\mathcal{S}$	Indexed set of stored models
$\|\mathcal{S}\|$	Number of stored models
k_i	Model i
$\phi_{t,k}$	Variational parameter after t iterations of model k
d_{k_i,k_j}	Distance between model k_i and model k_j
$\mathbf{A}$	Distance matrix
$p_k(\mathbf{y}\|\mathbf{x})$	Probability of the output (label) y for a given input data sample x
R	Number of Monte Carlo samples

CHAPTER 5

Forecasting Resources Utilization for Large-Scale Distributed Systems

5.1 Introduction

Federated learning systems often include thousands of clients [1] with very different hardware (i.e., CPU, memory), network connectivity (i.e., 4G, 5G, Wi-Fi), and battery level. Consequently, the effective management of such large-scale and heterogeneous distributed systems is very challenging.

In the previous chapters, we have seen that as a result of this heterogeneity, federated learning systems often have to face situations where clients drop out of the collaboration before the completion of the learning task due to connectivity or energy issues. Another consequence of device heterogeneity is that the server often receives updates from clients at different times asynchronously. Hence, a node with few available resources (i.e., computation, communication) can significantly slow down the whole federated learning process. This phenomenon is often referred to as the *straggler effect* and is the main bottleneck in realizing federated learning on a diverse network. A node with more data than others can also become the *straggler* as the former node will require a longer time to update its model, unless it has better computation resource than other clients [72].

To combat the straggler effect, some federated learning approaches use *active*

[1] The terms clients, nodes, devices are used interchangeably.

sampling techniques, which aim to train only on a subset of clients selected based on their ability to meet certain criteria (e.g., hardware characteristic, connectivity), which should ensure the training latency to be minimized. For example, authors in [73], select clients that are able to download, update, and upload ML models within a certain a time-window. To improve active sampling in federated learning but more generally for system management purposes such as optimizing job scheduling and learning process, it is very beneficial for the central server [2] to monitor and predict the resource conditions and utilization on the clients' side [1].

Several challenges exist for the central node to collect and forecast resource utilization at each client/machine in such a large-scale distributed system. First, it is often bandwidth-consuming and unnecessary to transmit all the measurements collected at local nodes to the central node. Second, predictive models for data forecasting typically have high complexity, thus running a forecasting model for the time-series measurement data collected at each local node would consume too much computational resource. Third, measurements at each local node are collected in an online manner, which form a time series; decisions related to data collection and forecasting need to be made in an online manner as well. Such online processing for large scale system in real time is very demanding.

In this chapter, we address the above challenges and propose a mechanism that efficiently collects and forecasts the resource utilization at machines/clients in a large-scale distributed system. The results provided by our mechanism can be further used for system management such as resource allocation. We focus on the collection and forecasting of resource utilization in this work, and leave its application to system management for future work. Our main contributions are summarized as follows.

1. We propose an algorithm for each local node to adaptively decide when to

[2] Also referred as the central node or central controller.

transmit its latest measurement to the central node, subject to a maximum frequency of transmissions that is given as a system-constraint parameter. The algorithm adapts to the degree of changes in observations since the last transmission, so that the allowed transmission bandwidth is efficiently used.

2. We propose a dynamic clustering algorithm for the central node to partition the measurements received from local nodes into a given number of clusters. The algorithm allows the clustering to evolve over time, and the cluster centroids are treated as a compressed representation of the dynamic observations of the large distributed system.
3. We propose a forecasting mechanism where the centroids of each cluster evolving over time constitute a time series that is used to train a forecasting model. The trained model is then used to forecast the future resource utilizations of a group of local nodes.
4. Extensive experiments of our proposed mechanism have been conducted using three real-world computing cluster datasets, to show the effectiveness of our proposed approach.

The clustering, model training, and forecasting are all performed in an online manner, based on "intermittent" measurement data received at the central node.

The rest of this chapter is organized as follows. In the next section, we review the related work. In Section 5.4, we present the system overview together with some definitions. In Section 5.3, we present several experiments to motivate this work. The proposed algorithms are described in Section 5.5. The experimentation settings and results are given in Section 5.6, and Section 5.7 draws our conclusion. The summary of notations used in this chapter is presented in Table 5.5.

5.2 Related Work

The existing body of work that uses prediction/forecasting models to assist resource scheduling mostly focuses on aggregated workloads or resource demands that can be described as a single time series [74, 75, 76, 77]. While these approaches are useful for predicting the future demand, they do not capture the dynamics of resource utilization at individual physical machines, and hence cannot predict how much resource is utilized or available in the physical system. In this work, we focus on resource utilization at machines in the large-scale distributed system, which is more complex because each machine generates a time-series measurement data on its own.

Some existing approaches for efficient data collection in a distributed system involve only a selected subset of local nodes that transmit data to the central node [78, 79, 80, 81, 82, 83, 84, 1]. More specifically, techniques in [83, 84, 1] select the best set of monitors (local nodes) subject to a constraint on the number of monitors, and infer data from the unobserved local nodes based on Gaussian models. Methods in [78, 79, 80, 81, 82] are based on compressed sensing, where a subset of local nodes is randomly selected to collect data, then matrix completion is applied to reconstruct data from unobserved nodes. The approaches based on compressed sensing generally perform worse than Gaussian-based approaches [1]. All these approaches where only some of the local nodes send data in the monitoring phase lead to unbalanced resource consumption (such as communication bandwidth and energy).

To avoid unbalanced resource consumption, some existing approaches consider settings where every node sends data to the central node but with a sampling rate adapted directly at each node [85, 86, 87, 88, 89]. However, the sampling rate in these methods is only implicitly related to the transmission frequency. None of them allows one to specify a target transmission frequency which is proportional to the limited, available communication bandwidth. In this work, we propose an algorithm that decides when to transmit subject to a maximum transmission frequency. This

allows the system to explicitly specify the communication budget.

For the clustering of local node measurements, Gaussian models are widely used, such as in [83, 84, 1]. However, these methods require a separate training phase to estimate the covariance matrix, during which it needs to collect all the data from all local nodes, which can be bandwidth consuming. In addition, a sufficiently large number of samples are required for a good estimation of the covariance matrix. When the correlation among local nodes vary frequently, which is the case with resource utilization at machines in distributed systems (see Section 5.3 for further discussion), the system may not be able to collect enough samples to estimate the covariance matrix with a reasonable accuracy. In this work, we propose a clustering mechanism that works well with highly varying resource utilization data.

The evolution of clusters over time is related to the area of evolutionary clustering [90, 91, 92, 93], for which typical applications include community matching in social science [92], disease diagnosis in bio-informatics [94], user preference modelling in dynamic recommender systems [95], etc. To our knowledge, evolutionary clustering techniques have not been applied to the dynamic clustering and forecasting of resource utilization at multiple machines, where the objectives are different from the above applications.

In summary, while there exist methods in the literature that are related to specific parts of our problem, they focus on different scenarios or applications and do not directly apply to our problem, as explained above. Furthermore, to our knowledge, a mechanism that efficiently collects and forecasts resource utilization of all machines in a distributed system does not exist in the literature. This chapter overcomes the challenge of developing such a mechanism with different components working smoothly together, while providing good performance in practical settings.

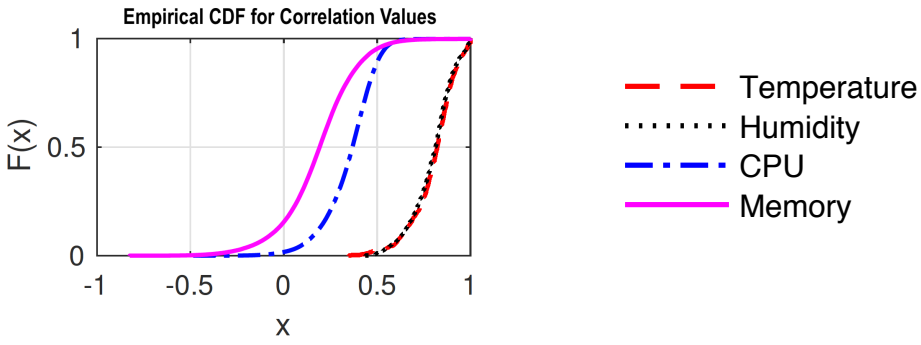


Figure 5.1: Empirical cumulative distribution function (CDF) of correlation values of different datasets.

5.3 Motivational Experiment

To illustrate the challenge in the problem we study, we start with a motivational experiment comparing the long-term spatial correlations[3] in resource utilizations at different machines in a distributed computing environment and sensor measurements at different nodes in a sensor network.

We consider the sensor network dataset collected by Intel Research Laboratory at Berkeley [96], which includes sensor measurements over 12 days, and the Google cluster usage trace (version 2) [97], which includes resource utilizations at machines over one month. The empirical cumulative distribution function (CDF) of the spatial correlation values computed on the temperature and humidity data from the sensor dataset and the CPU and memory utilization data (aggregated for all tasks running on each machine) from the Google cluster dataset are plotted in Fig. 5.1, where each type of data is considered separately.

We see that for CPU and memory utilization, most of the spatial correlation values are between -0.5 and 0.5, whereas most correlation values are above 0.5 for temperature and humidity data. This shows that in the long term (over the entire duration of the dataset), the spatial correlation in resource utilization among machines

[3]The (spatial) correlation of two nodes is defined as the sample covariance of measurements obtained at the two nodes divided by the standard deviations of both sets of measurements (each obtained at one of the two nodes) [77].

in a distributed computing system is much weaker than the spatial correlation in sensor measurements at different nodes in a sensor network. Therefore, we *do not* have strong long-term spatial correlation in our scenario, which is required by Gaussian-based methods for covariance matrix estimation (see also the related discussions in Section 5.2). Hence, Gaussian models which are widely used in the clustering of sensor network data [83, 84, 1] are not suitable for our case with resource utilization data. This justifies the need of developing a new clustering mechanism that focuses more on short-term spatial correlations.

More detailed comparison between our approach and the Gaussian-based approach in [1] will also be presented later in Section 5.6.5.

5.4 Definitions and System Overview

We consider a distributed system with N local nodes (machines) generating resource utilization measurements, and a central node (controller) that receives a summary of all the local measurements and forecasts the future. We assume that time is slotted. For each time step t, let $\mathbf{x}_t := [x_{1,t}, x_{2,t}, ..., x_{N,t}]$ denote the N-tuple that contains the true measurements of N local nodes and let $\mathbf{z}_t := [z_{1,t}, z_{2,t}, ..., z_{N,t}]$ be the measurements stored at the central node. Here, $x_{i,t}$ and $z_{i,t}$ $(1 \leq i \leq N)$ are d-dimensional vectors, where d is equal to the number of resource types (e.g., CPU, memory). The values in $\mathbf{z}_t$ depend on the transmission frequency (i.e., how often each local node sends its measurement to the central node). For each node i, let $\beta_{i,t}$ be an indication variable such that $\beta_{i,t} = 1$ if node i has sent its most recent measurement at time step t to the central node, otherwise $\beta_{i,t} = 0$. When $\beta_{i,t} = 0$,the central node sets $z_{i,t} = x_{i,t-p}$, where $p \geq 0$ is defined as the smallest p such that $\beta_{i,t-p} = 1$. If $\beta_{i,t} = 1$, then $p = 0$ and $z_{i,t} = x_{i,t}$.

We define K as a given input parameter to the system that specifies the number

of different forecasting models the system uses, which is related to the computational overhead. At each time step t, the central node partitions the N measurements $z_{1,t}, z_{2,t}, ..., z_{N,t}$ into K clusters, so that one forecasting model can be used for each cluster. Let $C_{j,t}$ ($1 \leq j \leq K$) denote the j-th cluster at time step t, which is defined as a set of indices of local nodes whose measurements are included in this cluster, i.e., $C_{j,t} \subseteq \{1, 2, ..., N\}$. Each cluster j has a centroid, defined as

$$c_{j,t} := \frac{1}{|C_{j,t}|} \sum_{i \in C_{j,t}} z_{i,t} \tag{5.1}$$

where $|\cdot|$ denotes the cardinality (size) of the set.

At time step t, a time-series forecasting model is trained using the time series formed by the set of historical centroids (i.e., $\{c_{j,\tau} : \tau \leq t\}$) for each cluster j. The model can forecast future values of the cluster centroid, i.e., for any forecasting step $h \geq 1$, the model provides a forecasted value $\hat{c}_{j,t+h}$ at the future time step $t + h$. The future resource utilization at each individual local node i is predicted as the value of its centroid plus an offset for this node, thus we define[4]

$$\hat{x}_{i,t+h} = \hat{c}_{j,t+h} + \hat{s}_{i,t+h} \tag{5.2}$$

for $i \in \hat{C}_{j,t+h}$, where $\hat{C}_{j,t+h}$ is the forecasted set of nodes in cluster j at time step $t + h$, and $\hat{s}_{i,t+h}$ is the forecasted offset of node i with respect to the centroid of cluster j (to which node i is forecasted to belong to) at time step $t+h$. In this way, the estimation of $\hat{x}_{i,t+h}$ involves both spatial estimation[5] (using cluster centroid and per-node offset as estimation of values for individual nodes) and temporal forecasting.

[4]For convenience (and with slight abuse of notation), we use the subscript $t + h$ to denote that the current time step is t and we forecast h steps ahead. With this notation, even if $t_1 + h_1 = t_2 + h_2$, we may have $\hat{x}_{i,t_1+h_1} \neq \hat{x}_{i,t_2+h_2}$ if $t_1 \neq t_2$.

[5]The use of the term *spatial* estimation or *spatial* correlation is for notional convenience. We acknowledge that the clustering behavior of the measurement data from different local nodes result from their spatial relationship as well as non-spatial reasons such as application-driven workloads.

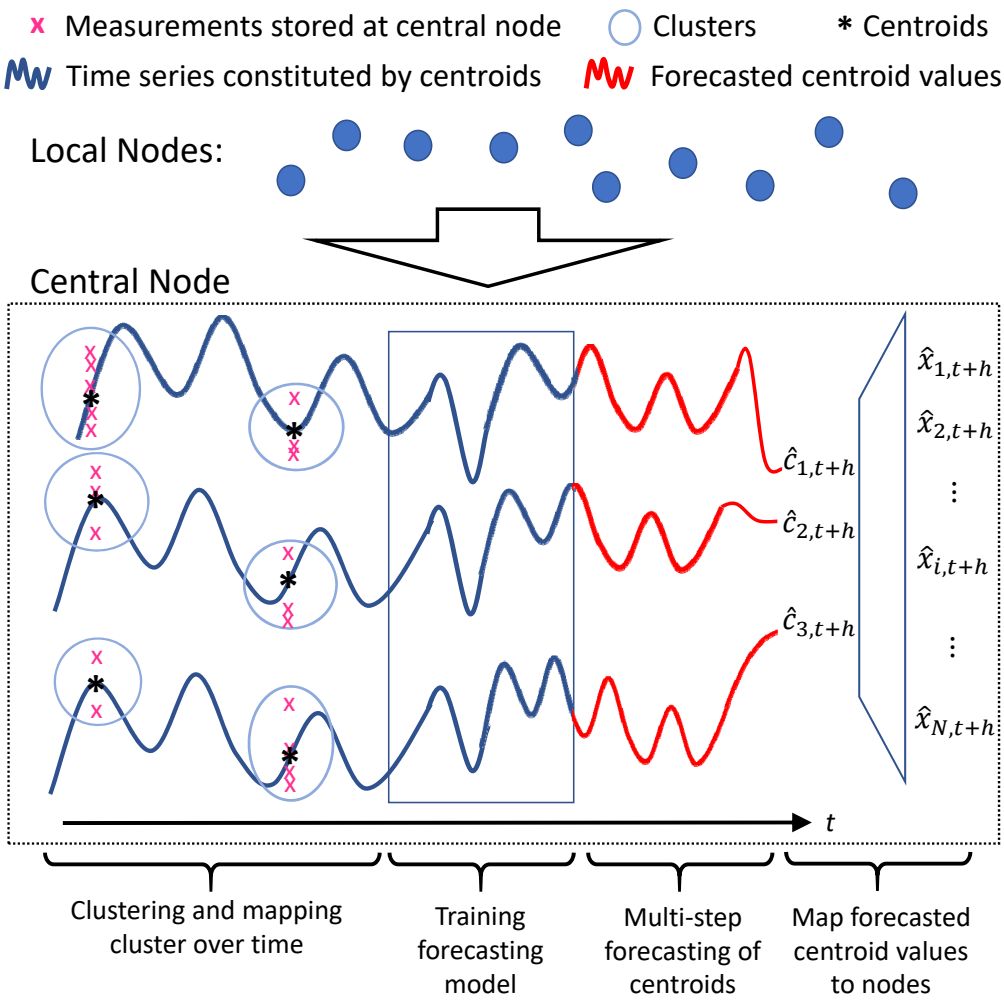


Figure 5.2: System overview.

Fig. 5.2 illustrates the system with the functionalities described above.

We define the root mean square error (RMSE) of $\hat{\mathbf{x}}_{t+h} := [\hat{x}_{1,t+h}, \hat{x}_{2,t+h}, ..., \hat{x}_{N,t+h}]$ for $h \geq 0$ as

$$\mathrm{RMSE}(t,h) := \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\hat{x}_{i,t+h} - x_{i,t+h}\|^2} \tag{5.3}$$

where we define $\hat{x}_{i,t} := z_{i,t}$ for $h = 0$ for convenience. With this definition, when $h = 0$, the RMSE only includes the error caused by infrequent transmission of local node measurements to the central node. We also note that the true value $\mathbf{x}_{t+h}$ cannot be observed directly by the central node.

We also define the time-averaged RMSE over T time steps for a given forecasting step h as

$$\overline{\mathrm{RMSE}}(T,h) := \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\mathrm{RMSE}(t,h))^2} \tag{5.4}$$

where the time average is over the square error and the square root is taken afterwards.

Let B_i ($0 \leq B_i \leq 1$) denote the maximum transmission frequency (for node i). Using the above definitions, and considering a maximum forecasting range H, the algorithms to be introduced in the next section aim at solving the following problem:

$$\begin{aligned} \min \quad & \lim_{T\to\infty} \sqrt{\frac{1}{H+1}\sum_{h=0}^{H}(\overline{\mathrm{RMSE}}(T,h))^2} \\ \text{s.t.} \quad & \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\beta_{i,t} \leq B_i, \quad \forall i \end{aligned} \tag{5.5}$$

where the minimization is over all $\{\beta_{i,t}\}$, $\{C_{j,t}\}$, $\{\hat{C}_{j,t+h}\}$, $\{\hat{c}_{j,t+h}\}$, and $\{\hat{s}_{i,t+h}\}$. Intuitively, we would like to find the transmission schedule (indicator) $\beta_{i,t}$ for each local node i and time step t, the membership of clusters $C_{j,t}, \forall j$ for each time step t, and the forecasted cluster memberships, centroids, and offsets for every forecasting step $h \in [0, H]$ computed at each time step t, to minimize the average RMSE over all forecasting steps and all time steps.

As we do not make any assumption on the characteristics of the time series constituting the cluster centroids $\{c_{j,t}\}$, we cannot hope to find the theoretically optimal forecasting scheme due to the complex system dynamics. In addition, it is often reasonable in the clustering step to minimize the error between the data and their closest cluster centroids (we refer to this error as the "intermediate RMSE" later in the chapter), which is the K-means clustering problem and is NP-hard [98]. We also note that an online algorithm is required because measurements from local nodes are obtained over time and decisions have to be made only based on the current and

past information (with future information unknown to the algorithm). All the above impose challenges in solving (5.5). Consequently, we propose online heuristics to solve the problem (5.5) approximately in the next section. These heuristics work well in practice as we show in Section 5.6 later.

5.5 Proposed Algorithms

5.5.1 Measurement Collection with Adaptive Transmission

In every time step t, each node i determines its transmission action $\beta_{i,t}$, i.e., whether it transmits its current measurement $x_{i,t}$ to the central node or not. To capture the error of the measurements stored at the central node, we define a *penalty function*

$$F_{i,t}\left(\beta_{i,t}\right) := \begin{cases} \frac{1}{d}\left\|z_{i,t} - x_{i,t}\right\|^2, & \text{if } \beta_{i,t} = 0 \\ 0, & \text{if } \beta_{i,t} = 1 \end{cases}. \tag{5.6}$$

To take into account the maximum transmission frequency B_i, we also define $Y_i\left(\beta_{i,t}\right) := \beta_{i,t} - B_i$. We also define $V_0 > 0$ and $\gamma \in (0,1)$ as a control parameters. The algorithm that runs at each node i to determine $\beta_{i,t}$ is given as follows.

1. In the first time slot $t = 1$, initialize a variable $Q_i(t) \leftarrow 0$. The variable $Q_i(t)$ represents the length of a "virtual queue" at node i.
2. For every $t \in \{1, 2, 3, ...\}$, choose $\beta_{i,t}$ according to

$$\beta_{i,t} \leftarrow \arg\min_{\beta \in \{0,1\}} V_t F_{i,t}(\beta) + Q_i(t) Y_i(\beta) \tag{5.7}$$

where

$$V_t := V_0 \cdot (t+1)^{\gamma}. \tag{5.8}$$

Then, update the virtual queue length according to

$$Q_i(t+1) \leftarrow Q_i(t) + Y_i(\beta_{i,t}). \tag{5.9}$$

The intuition behind the above algorithm is as follows. The virtual queue length $Q_i(t)$ captures how much the B_i constraint in (5.5) has been violated up to the current time step t. The determination of $\beta_{i,t}$ in (5.7) considers a trade-off between the penalty (error) $F_{i,t}(\beta)$ and constraint violation (related to $Q_i(t)$), where the trade-off is controlled by the parameter V_t. When $Q_i(t)$ is large, the term $Q_i(t)Y_i(\beta)$ in (5.7) becomes dominant, and the algorithm tends to choose $\beta = 0$ because this gives a negative value of $Q_i(t)Y_i(\beta)$ which is in favor of the minimization. Since $\beta = 0$ corresponds to not transmitting, this relieves the constraint violation. When $Q_i(t)$ is small and $\|z_{i,t} - x_{i,t}\|^2$ is relatively large, the term $V_t F_{i,t}(\beta)$ in (5.7) is dominant. In this case, the algorithm tends to choose $\beta = 1$ because this will make $F_{i,t}(\beta) = 0$ and reduces the error of measurements stored at the central node.

The above algorithm is a form of the drift-plus-penalty framework in Lyapunov optimization [99]. According to Lyapunov optimization theory, as long as $F_{i,t}(\beta)$ has a finite upper bound[6], the above algorithm can always guarantee that the B_i constraint in (5.5) is satisfied with equality (for $T \to \infty$ as given in the constraint, not necessarily for finite T), because $\lim_{t\to\infty} Q_i(t)/t = 0$ (see [99, Chapter 4]). Note that satisfying the B_i constraint with equality is always not worse than satisfying it with inequality, because more transmissions cannot hurt the RMSE performance. For finite T, the satisfaction of the B_i constraint is related to the parameter V_t, which can be tuned by parameters V_0 and γ. From (5.8), we see that V_t increases with t, which means that we give more emphasis on minimizing the penalty function when t is large. This is because for a larger t, we can allow a larger $Q_i(t)$ while still

[6] $F_{i,t}(\beta)$ usually has a finite upper bound because measurement data is usually finite. Also note that the lower bound of $F_{i,t}(\beta)$ is zero thus finite.

maintaining $Q_i(t)/t$ close to zero.

Note, however, that the penalty function $F_{i,t}(\beta)$ depends on transmission decisions in previous time steps that impact the value of $z_{i,t}$. Therefore, the optimality analysis of Lyapunov optimization theory does not hold for our algorithm, and we do not have a theoretical bound on how optimal the result is. Nevertheless, we have observed that this algorithm with the current penalty definition works well in practice (see experimentation results in Section 5.6).

5.5.2 Dynamic Cluster Construction Over Time

We now discuss how the central node computes the clusters $C_{j,t}$, for $1 \leq j \leq K$, from $\mathbf{z}_t$ over time. The computation includes two steps. First, K-means clustering is computed using the stored measurements $\mathbf{z}_t$ in time step t only. Second, the clusters computed in the first step are re-indexed so that they align the best with the clusters computed in previous time steps. The re-indexing step is only performed for $t > 1$.

The first step of K-means clustering is straightforward and efficient heuristic algorithms for K-means exist [100]. Let $C'_{k,t}$ $(1 \leq k \leq K)$ denote the K-means clustering result on $\mathbf{z}_t$ in time step t. If $t = 1$, we let $j = k$, such that $C_{j,t} = C'_{j,t}, \forall j$, where we recall that $\{C_{j,t} : \forall j\}$ is the final set of clusters in time step t. If $t > 1$, the cluster indices of $\{C'_{k,t} : \forall k\}$ need to be reassigned in order to obtain $\{C_{j,t} : \forall j\}$, because the cluster indices resulting from the K-means algorithm is random, and for each cluster $C'_{k,t}$, we need to find out which cluster among $\{C_{j,t-1} : \forall j\}$ in the previous time step $t-1$ it evolves from.

To associate the clusters $\{C'_{k,t} : \forall k\}$ in time step t with the clusters in previous time steps, we define a similarity measure between the k-th cluster from the K-means result in time step t, i.e., $C'_{k,t}$, and the j-th clusters in a subset of previous time steps.

Formally, the similarity measure is defined as

$$w_{k,j} = \left| C'_{k,t} \cap \left(\bigcap_{m=1}^{\min\{M,t-1\}} C_{j,t-m} \right) \right| \tag{5.10}$$

where $M \geq 1$ specifies the number of time steps to look back into the history when computing the intersection in the similarity measure. Intuitively, the similarity measure $w_{k,j}$ specifies how many local nodes exist concurrently in the k-th cluster obtained from the K-means algorithm in time step t and in the j-th clusters in all M most recent time steps (excluding time step t). If $w_{k,j}$ is large, it means that most of the nodes in the corresponding clusters are the same.

Now, to find $C_{j,t}$ from $C'_{k,t}$, we find a one-to-one mapping between the indices j and k. Let φ denote the one-to-one mapping from k to j. We would like to find the mapping φ such that the sum similarity is maximized, i.e.,

$$\max_{\varphi} \sum_{k=1}^{K} w_{k,\varphi(k)}. \tag{5.11}$$

Intuitively, with the mapping φ found from (5.11), the clusters $\{C_{j,t} : \forall j\}$ are indexed in such a way that most nodes remain in the same cluster in the current time step t and M previous time steps. In this way, the evolution of the centroids of each cluster j represents a majority of local nodes within that cluster, and it is reasonable to perform time-series forecasting with the centroids of clusters that are dynamically constructed in this way.

Solution to (5.11): The problem in (5.11) is equivalent to a maximum weighted bipartite graph matching problem, where one side of the bipartite graph has nodes representing the values of k, the other side of the bipartite graph has nodes representing the values of j, and each k-j pair is connected with an edge with weight $w_{k,j}$. This can then be solved in polynomial time using existing algorithms for maximum

weighted bipartite graph matching, such as the Hungarian algorithm [101].

The parameter M in the similarity measure (5.10) controls whether to consider long or short term history when computing the similarity. The proper choice of M is related to the temporal variation in the data correlation among different local nodes, because each cluster contains a group of nodes that are (positively) correlated with each other. Our experimentation results in Section 5.6 show that a fixed value of M usually works well for a given scenario.

Our clustering approach can be extended in several ways. For example, one can define a time window of a given length, which contains multiple time steps, and perform clustering on extended feature vectors that include measurements at multiple time steps within each time window [102]. In this case, t represents the time window index, and everything else in our approach presented above works in the same way. We mainly focus on dynamic settings where the time series and node correlation can fluctuate frequently. In such settings, as we will see in the experimentation results in Section 5.6, it is best to use a time window of length one (equivalent to no windowing), so that the clustering can adapt to the most recent measurements. We can also perform clustering on each type of resource (e.g., CPU, memory) independently from other resource types, in which case the K-means step is performed on one-dimensional vectors (equivalent to scalars). We will see in Section 5.6 that this way of independent clustering performs better than joint clustering on the datasets we use for evaluation.

Our dynamic clustering approach shares some similarities with the approach in [92]. However, we define a different similarity measure that can look back multiple time steps and is not normalized. This is more suitable for the RMSE objective in (5.5) which considers the errors at all nodes. Moreover, we focus on the clustering and forecasting of time-series data which is different from existing work.

5.5.3 Temporal Forecasting

As discussed in Section 5.4, temporal forecasting is performed using models trained on historical centroids of measurements stored at the central controller. The models can include Autoregressive Integrated Moving Average (ARIMA) [103], Long Short-Term Memory (LSTM) [104], etc. Different models have different computational complexities. When the system starts for the first time, there is an initial data collection phase where there is no forecasting model available to use. Afterwards, forecasting models are trained on the time-series constituted by the historical centroids of clusters. After the models are trained, the system can forecast future centroids using the models, based on the most updated measurements at the central node. The transient state of each model gets updated whenever a new measurement is available. The models are retrained periodically at a given time interval using all (or a subset of) the historical cluster centroids up to the current time.

As explained in Section 5.4, at time step t, the forecasted resource utilization at node i in the future time step $t+h$ is computed using the forecasted centroid plus an offset, i.e., $\hat{x}_{i,t+h} = \hat{c}_{j,t+h} + \hat{s}_{i,t+h}$ where j is chosen such that $i \in \hat{C}_{j,t+h}$. We explain how to find the forecasted cluster $\hat{C}_{j,t+h}$ and the offset $\hat{s}_{i,t+h}$ in the following. We define M' as the number of time steps to look back into the history (excluding the current time step t). For each node i, consider the time steps within the interval $[t - M', t]$, and compute the frequency that node i belongs to the j-th cluster $C_{j,t}$ within this time interval, for all j. Let j^* denote the cluster that node i belongs to *for the most time* within $[t - M', t]$. The algorithm then predicts that node i belongs to the j^*-th cluster in time step $t + h$. By finding j^* for all i, the forecasted cluster $\hat{C}_{j,t+h}$ is obtained for all j.

For node $i \in \hat{C}_{j,t+h}$, the offset $\hat{s}_{i,t+h}$ is computed as

$$\hat{s}_{i,t+h} = \frac{1}{M'+1} \sum_{m=0}^{M'} \alpha_{t-m}(z_{i,t-m} - c_{j,t-m}) \tag{5.12}$$

where $\alpha_{t-m} \in (0, 1]$ is a scaling coefficient that ensures the cluster centroid plus the offset $c_{j,t-m} + \alpha_{t-m}(z_{i,t-m} - c_{j,t-m})$ still belongs to cluster j in time step $t-m$, i.e., its value is still closest to the centroid $c_{j,t-m}$ of cluster j compared to the centroids of all other clusters. If $z_{i,t-m}$ belongs cluster j, we choose $\alpha_{t-m} = 1$. Otherwise, we choose α_{t-m} as the largest value so that $c_{j,t-m} + \alpha_{t-m}(z_{i,t-m} - c_{j,t-m})$ belongs to cluster j. This is useful because we do not want the offset to be so large that the resulting estimated value belongs to a different cluster (other than cluster j), as the forecasted $\hat{x}_{i,t+h}$ is still based on the forecasted centroid $\hat{c}_{j,t+h}$ of cluster j.

5.6 Experimentation Results

5.6.1 Setup

5.6.1.1 Datasets

We evaluate the performance of our proposed approach on three real-world computing cluster datasets. The first dataset is the *Alibaba cluster trace (version 2018)* [105] that includes CPU and memory utilizations of $4,000$ machines over a period of 8 days. The raw measurements are collected at 1 minute intervals (i.e., each local node obtains a new measurement every minute) and the entire compressed dataset is about 48 GB. The second dataset is the Rnd trace of the *GWA-T-12 Bitbrains dataset* [106]. It contains 500 machines, the data is collected over a period of 3 months (we only use data in the first month because there is a 24-hour gap between different months), and raw measurements are sampled at 5 minute intervals. The size of the dataset is 156 MB. The third dataset is the *Google cluster usage trace (version 2)* [97],

which contains job/task usage information of approximately $12,478$ machines[7] over 29 days, sampled at 5 minute intervals. The total size of the compressed dataset is approximately 41 GB. For each dataset, we pre-processed the raw data to obtain the *normalized* CPU and memory utilizations for each individual machine.

5.6.1.2 Choice of Parameters

Unless otherwise specified, we set the transmission frequency constraint $B_i = B := 0.3$ for all i, the control parameters for adaptive transmission $V_0 = 10^{-12}$ and $\gamma = 0.65$, the number of forecasting models (which is equal to the number of clusters) $K = 3$, the look-back durations for the similarity measure $M = 1$ and temporal forecasting $M' = 5$. The clustering is performed on the scalar values of the measurements of each resource type, unless noted otherwise. These parameter choices are justified in our experiments, which will be further discussed later in this section.

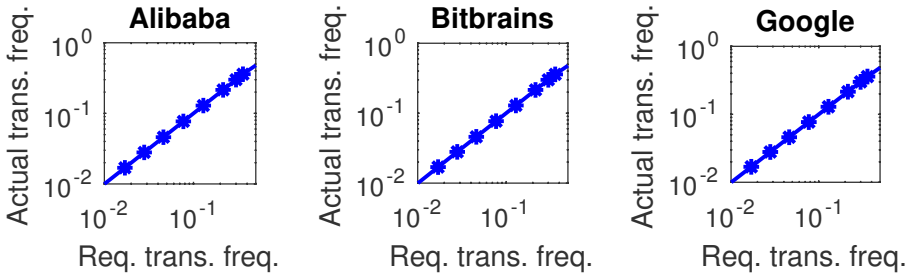


Figure 5.3: Behavior of the adaptive transmission algorithm

[7] We had to remove 2 machines that have error in the measurement data (unreasonably high CPU/memory utilization).

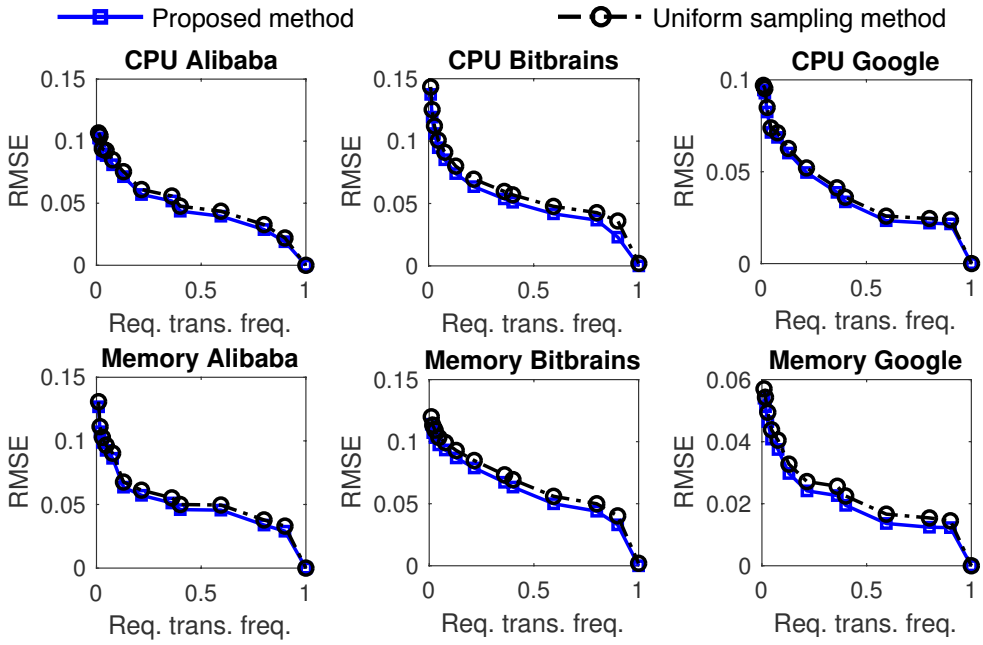


Figure 5.4: RMSE comparison of our proposed adaptive transmission method with the uniform sampling method

5.6.1.3 Forecasting Models

We use ARIMA and LSTM models for temporal forecasting. For the ARIMA model, after making some initial observations of the stationarity, auto correlation, and partial auto correlation functions, we conduct a grid search over the following ranges of parameters: the order of the auto-regressive terms $p \in [0, 5]$, the degree of differencing $d \in [0, 2]$, the order of the moving average terms $q \in [0, 5]$, and for the corresponding seasonal components: $P \in [0, 2]$, $D \in [0, 1]$, $Q \in [0, 2]$. The best model is selected from the grid search using the Akaike information criterion with correction term (AICc) [107]. For the LSTM model, we stack two LSTM layers, and on top of that we stack a dense layer with a rectified linear unit (ReLU) as activation function. Due to the randomness of LSTM, we plot the average forecasting results over 10 different simulation runs.

For both ARIMA and LSTM, the initial data collection phase includes the first 1000 time steps. Then, the models are retrained every 288 time steps, equivalent to a

day when the raw measurements are sampled at 5 minute intervals. For each cluster j, a separated model is trained for forecasting the centroids of this cluster. At every time step t, forecasting is made for a given number of time steps h ahead.

We present results on different aspects of our proposed mechanism in the following.

Remark: As mentioned in Section 5.2, to the best of our knowledge, there does not exist work in the literature that solves the entire problem in our setting. Therefore, we cannot compare our overall method with another existing approach. We will compare individual parts of our method with existing work where possible.

5.6.2 Adaptive Transmission Algorithm

We first study some behavior of the algorithm presented in Section 5.5.1. Fig. 5.3 shows that the required transmission frequency B always matches closely with the actual transmission frequency (with parameters V_0 and γ chosen as described in Section 5.6.1.2). This confirms that the algorithm is able to adapt the transmission frequency to remain within the B_i-constraint in (5.5).

In Fig. 5.4, we compare our proposed adaptive transmission approach with a uniform sampling approach, and show the time-averaged RMSE as defined in (5.4) with $h = 0$ and T equal to the total number of time steps in the dataset (recall that we defined $\hat{x}_{i,t} := z_{i,t}$ for $h = 0$, so the RMSE only includes error caused by infrequent transmission in this case). The *uniform sampling* baseline transmits each local node's measurement at a fixed interval, so that the average transmission frequency at each node i is equal to B_i. We see that our proposed approach outperforms the uniform sampling approach for any required transmission frequency. When the required transmission frequency is 1.0, we always have $z_{i,t} = x_{i,t}$ and the RMSE is zero for both approaches.

Even if our adaptive transmission method outperforms only slightly the uniform

sampling approach, we believe that it boosts the performance of the forecasting algorithm (later part of the framework). Indeed, by using our adaptive transmission algorithm, we make sure that the central server is notified of major measurement's change. These major changes are important as they determine cluster memberships and consequently have an influence on our forecasting algorithm. In other words, by only sending random sampled measurements, the central servers might miss important changes observed at a node and cluster this node in a wrong group, which can negatively impact the final predictions.

5.6.3 Spatial Estimation without Per-node Offset

In this subsection, we evaluate the impact of using cluster centroids to represent the group of nodes in the cluster, where we *ignore the offset* $\hat{s}_{i,t+h}$ and choose $h = 0$. We evaluate the *intermediate RMSE* which is the time-averaged RMSE between the data and their closest cluster centroids. This evaluation is useful because the forecasting models are trained on cluster centroids, so we would like the cluster centroids to be not too far from the actual measurements at each node even if there is no per-node offset added to the estimated value. It also provides useful insights on the clustering mechanism.

5.6.3.1 Impact of Clustering Dimensions

We first discuss the impact of different dimensions we cluster over time and over resource types. As mentioned in Section 5.5.2, we can cluster either on the measurement obtained at a single time step or multiple time steps, i.e., over different temporal dimensions. Fig. 5.5 shows the results of intermediate RMSE when we vary the temporal clustering dimension, where we cluster CPU and memory measurements separately and independently. We see that using a temporal clustering dimension of 1 (i.e., clustering the measurements obtained at a single time step) always gives the

Table 5.1: Intermediate RMSE of clustering independent scalars & full vectors

Resource type & dataset	Scalar	Full
CPU Alibaba	0.069	0.075
Memory Alibaba	0.066	0.072
CPU Bitbrains	0.086	0.089
Memory Bitbrains	0.096	0.098
CPU Google	0.063	0.082
Memory Google	0.055	0.067

best performance.

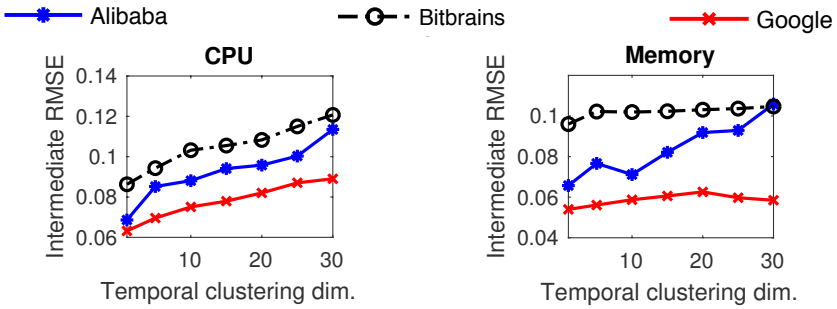


Figure 5.5: Intermediate RMSE of clustering different temporal dimensions.

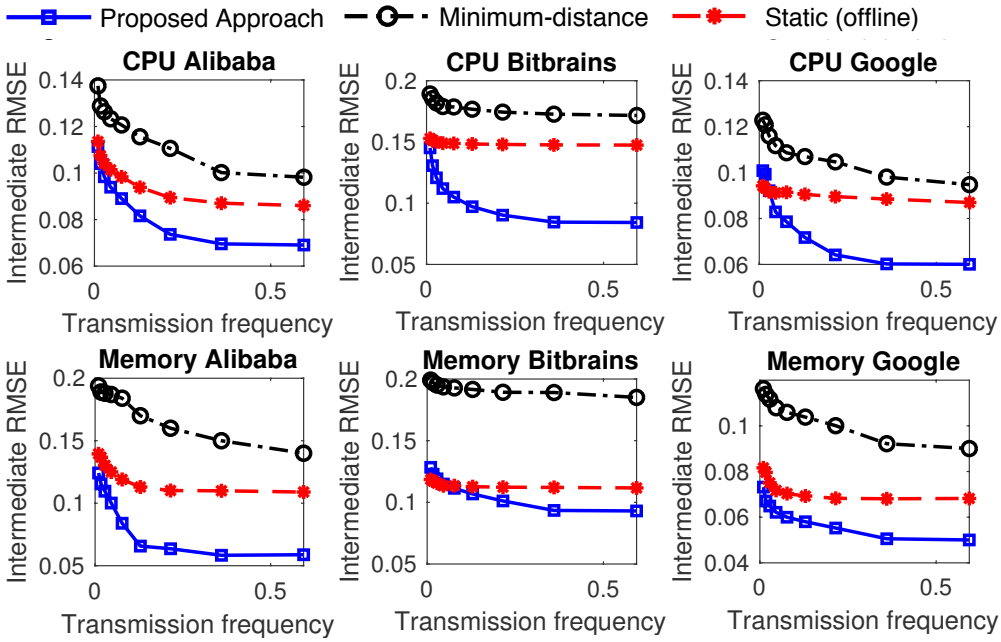


Figure 5.6: Intermediate RMSE when varying the transmission frequency B and fixing $K = 3$.

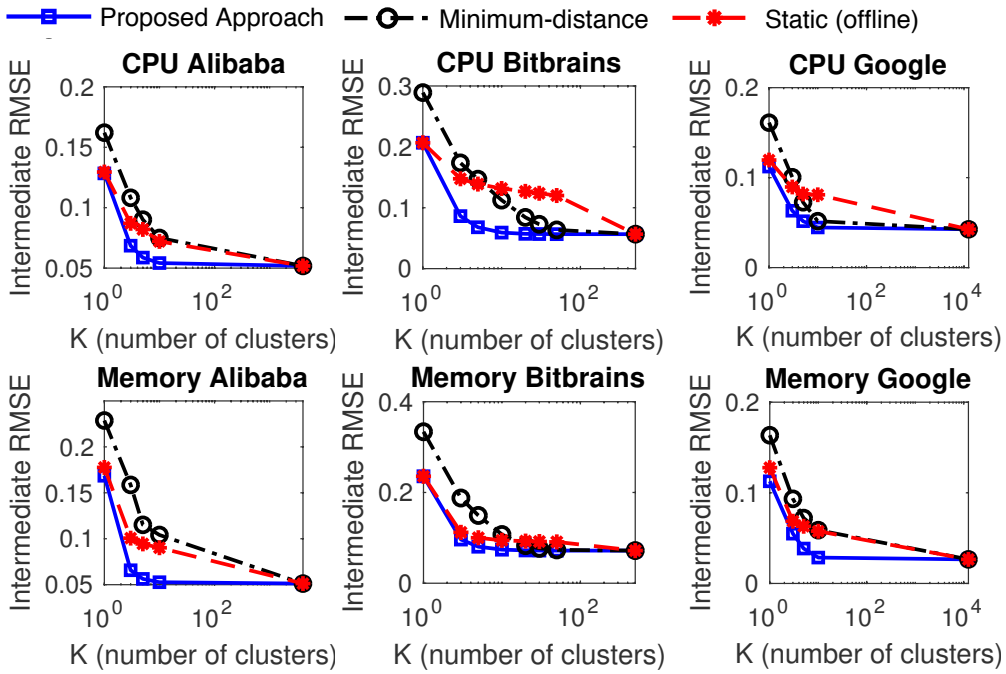
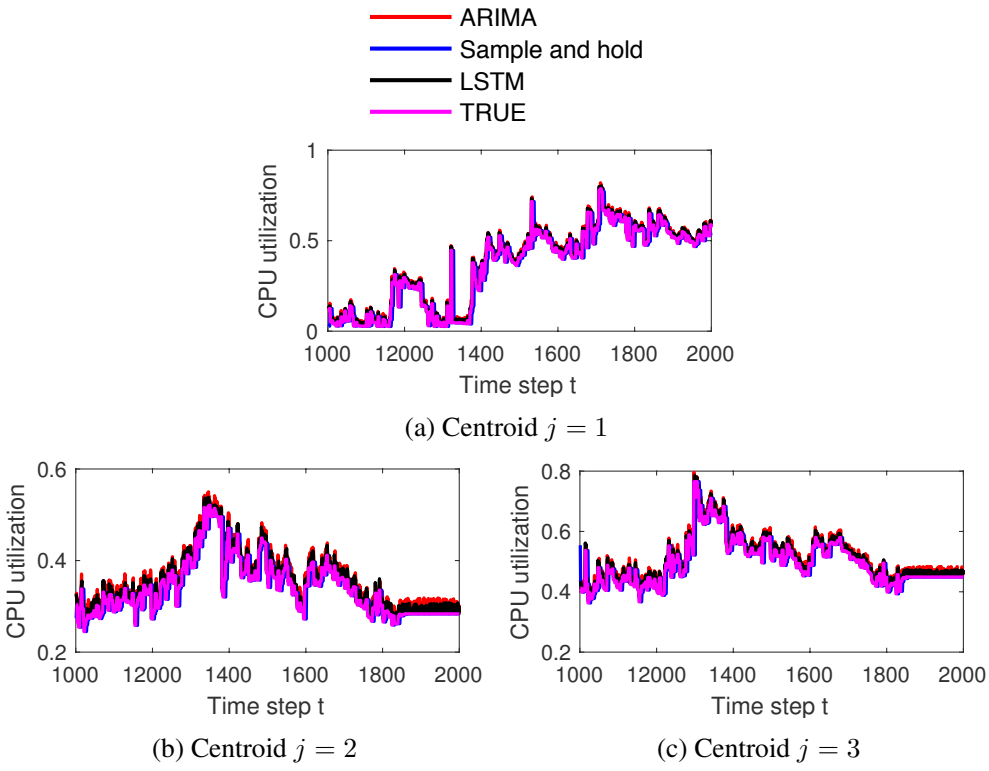


Figure 5.7: Intermediate RMSE when varying the number of clusters K and fixing $B = 0.3$.



(a) Centroid $j = 1$

(b) Centroid $j = 2$

(c) Centroid $j = 3$

Figure 5.8: Instantaneous true and forecasted ($h = 5$) results of $K = 3$ centroids on CPU data of Alibaba dataset.

Section 5.5.2 also mentions that we can either cluster different resource types independently using their scalar values, or we can jointly cluster vectors of multiple resource types. Table 5.1 compares the intermediate RMSEs of these two approaches, where the intermediate RMSEs are always computed for individual resource types, but the clustering is computed either on independent scalars or full vectors. We see that clustering using scalar values of each resource type performs better than clustering using the full vector. This suggests that the correlation among different types of resources in each dataset is relatively weak.

The above results show that it is beneficial to use scalar measurement values of each resource type at a single time step for clustering. We will use this setting in all our experiments presented next.

5.6.3.2 Different Clustering Methods

We compare our proposed dynamic clustering approach with two baselines. The first baseline *static clustering* is an *offline* baseline, where nodes are grouped into static clusters based on the entire time series at each node that is assumed to be known in advance. The clusters are found using K-means on multi-dimensional vectors, where each vector represents the entire time series at a node. With this setting, the clusters remain fixed over all time steps. The second baseline *minimum distance* is obtained by randomly selecting K nodes at each time step, treating the selected nodes as "centroids" and mapping the remaining nodes to the "centroids" based on minimum Euclidean distance between measurements. The minimum distance baseline represents approaches which select monitoring nodes randomly, such as [78, 79, 80, 81, 82].

Fig. 5.6 shows the intermediate RMSE with varying B while fixing $K = 3$. We can see that our proposed approach performs better than baseline approaches in (almost) all cases. Note that the static approach is an offline baseline with stronger assumptions than our proposed online approach. We also see that in most cases, the

intermediate RMSE starts to converge at approximately $B = 0.3$. This shows that a transmission frequency higher than 0.3 will not provide much benefit.

Fig. 5.7 shows the results with varying K while fixing $B = 0.3$. We see that the intermediate RMSE of the proposed approach is close to the lowest value even with only a few clusters (i.e., small value of K). This is a strong result because it shows that a small number of cluster centroids is sufficient for representing a large number of nodes. We also note that because $B = 0.3$, the measurements stored at the central node are not always up-to-date, which explains why the intermediate RMSE is larger than zero even when $K = N$.

The above observations explain the rationale behind choosing $B = 0.3$ and $K = 3$ as default parameters as mentioned in Section 5.6.1.2. In general, we can conclude that our proposed approach can provide close to optimal clustering error by using a small transmission frequency and a very small number of clusters, which significantly reduces the communication and computation overhead for system monitoring.

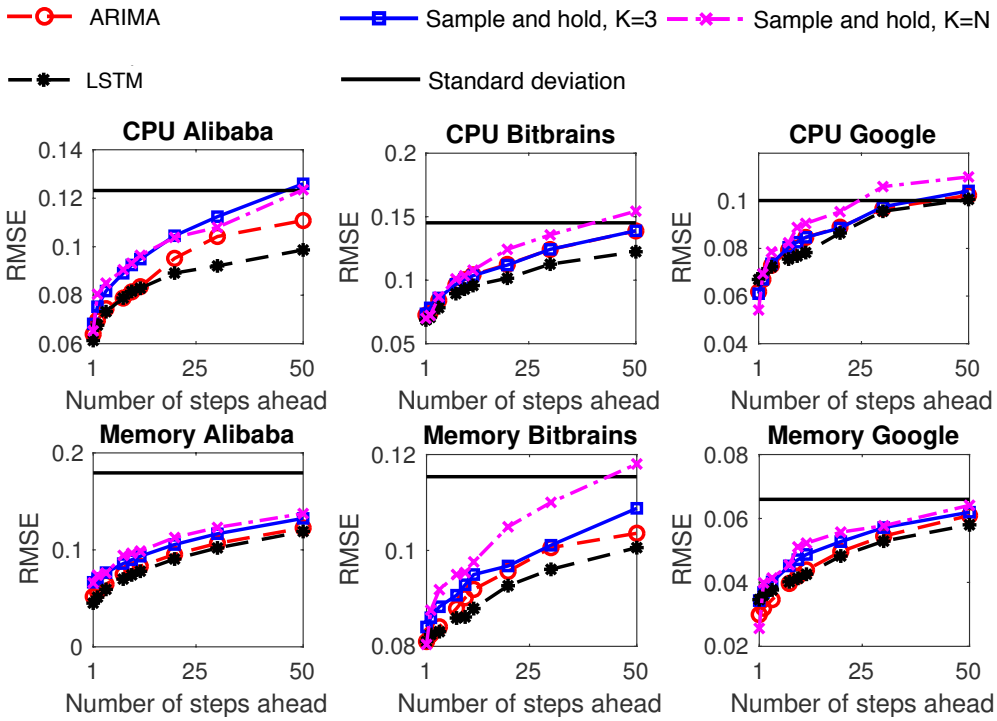


Figure 5.9: Time-averaged RMSE with different number of forecasting steps (h), with our proposed dynamic clustering approach.

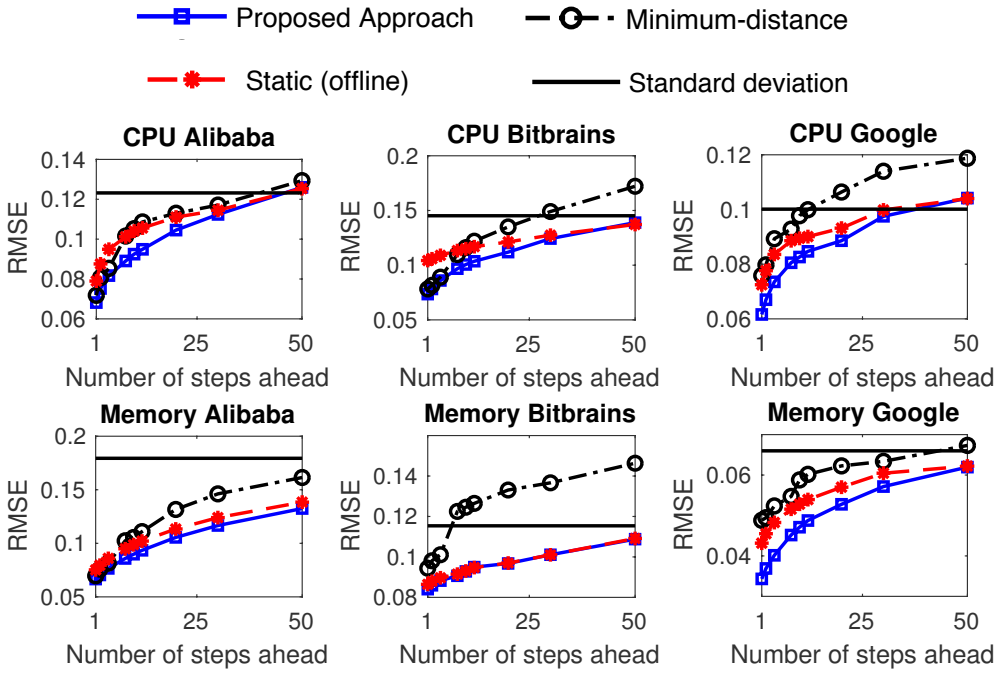


Figure 5.10: Time-averaged RMSE with different number of forecasting steps (h) using the sample-and-hold method.

5.6.4 Joint Spatial Estimation and Temporal Forecasting (with Per-node Offset)

We now consider the entire pipeline with joint spatial estimation (through dynamic clustering) and temporal forecasting. We include the per-node offset $\hat{s}_{i,t+h}$ in this subsection and focus on the time-averaged RMSE as defined in (5.4).

5.6.4.1 Different Forecasting Models

We compare our predictions based on ARIMA and LSTM with a *sample-and-hold* prediction method, which simply uses the cluster centroid values at time step t as the predicted future values. We also compare with the *standard deviation* computed over all resource utilizations over time (except for the instantaneous plot in Fig. 5.8). The standard deviation serves as an *error upper bound of an offline mechanism* where forecasting is made only based on long-term statistics (such as mean value) without considering temporal correlation.

We first show the instantaneous true and forecasted CPU utilization values of three different centroids for $t \in [1000, 2000]$ with the Alibaba dataset in Fig. 5.8, where the forecasting is for $h = 5$ steps ahead. We see that with our methods, the trajectories of the forecasted centroid values by all models follow very closely to that of the true centroid values.

The time-averaged RMSE with different forecasting models is shown in Fig. 5.9, where we include results for both $K = 3$ and $K = N$ for the sample-and-hold method, and use the default $K = 3$ for all the other methods. Also note that the standard deviation does not depend on K. We see that although sample-and-hold is simple enough to run on every local node (i.e., $K = N$), the case with $K = N$ generally performs worse than cases with $K = 3$. This is due to the fluctuation of resource utilization at individual nodes, which makes the forecasting model perform

Table 5.2: Aggregated training time (in seconds) of forecasting model on one centroid over the entire duration of the dataset

Dataset	ARIMA	LSTM
Alibaba data set (11519 total time steps)	61.25	855.34
Bitbrains data set (8259 total time steps)	33.4	554.97
Google data set (8350 total time steps)	37.86	554.97

badly when running on every node. The cluster centroids are averages of data at multiple nodes, which remove noisy fluctuations and provide better performance. LSTM performs the best among all the models, which is expected since LSTM is the most complex and advanced model compared to the others. We also see that the RMSE is lower than the standard deviation for most forecasting models when the forecasting step $h \leq 50$. This shows that our forecasting mechanism, which takes into account both spatial and temporal correlations, is beneficial over mechanisms that are only based on long-term statistics.

Table 5.2 shows the total (aggregated) computation time used for training the ARIMA and LSTM models for the entire duration of one centroid, on a regular personal computer (without GPU) with Intel Core i7-6700 3.4 GHz CPU, 16 GB memory. The model is trained or re-trained at each of the initial training and re-training periods defined in Section 5.6.1.3, and the result shown in Table 5.2 is the sum computation time for training at all periods. We can see that for data traces that span over at least multiple days, the total computation time used for model training is only a few minutes. Since we only need to train $K = 3$ models, the computation overhead (time) for training forecasting models is very small compared to the entire monitoring duration.

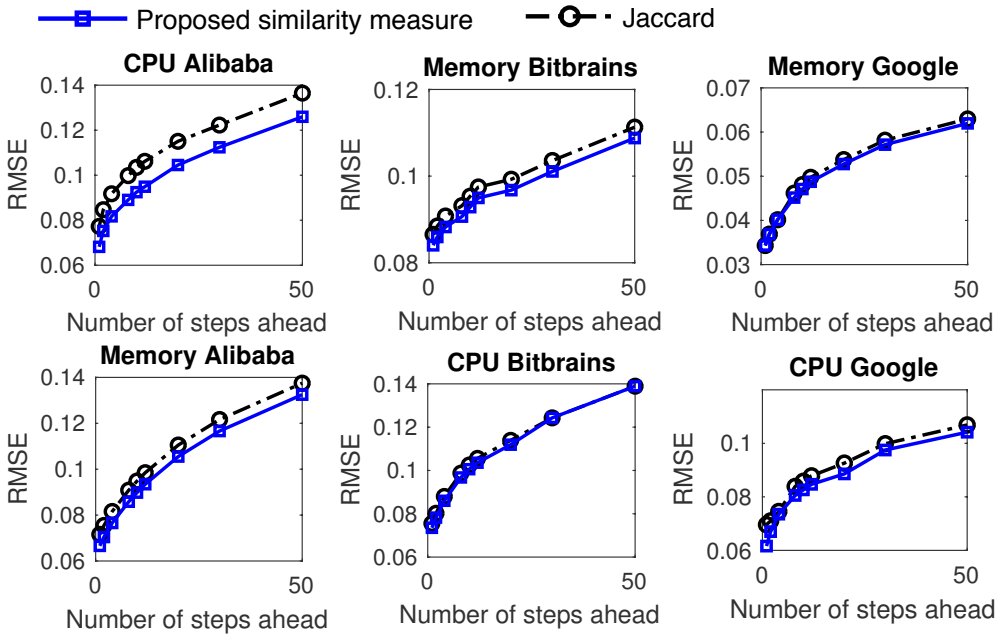


Figure 5.11: Time-averaged RMSE with Jaccard Index and our proposed similarity measure.

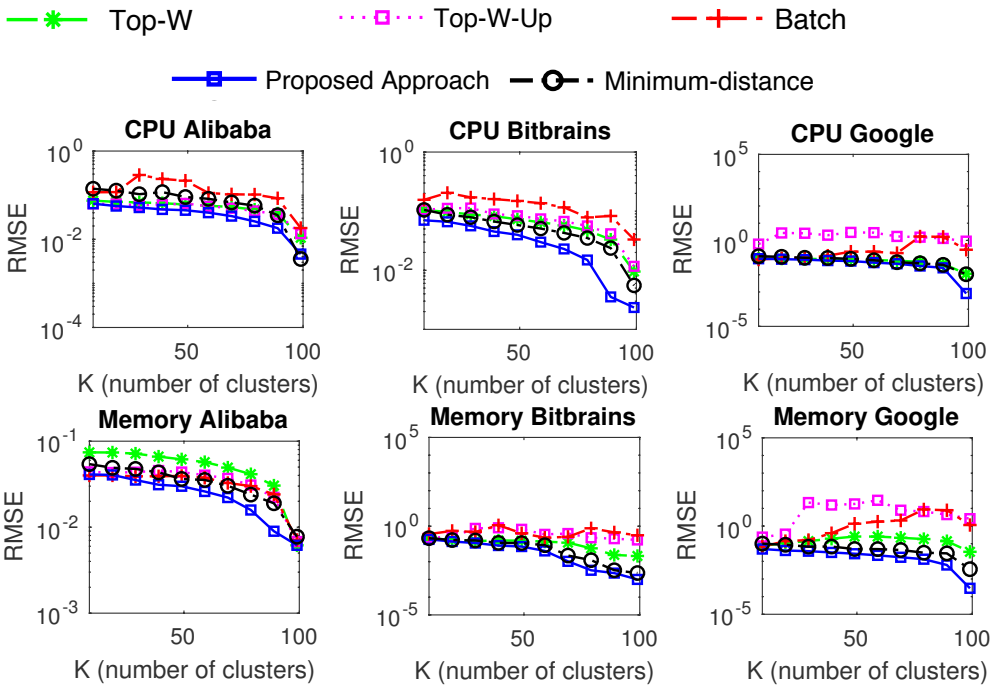


Figure 5.12: RMSE for comparison with [1] with different number of clusters (K).

Table 5.3: RMSE with Different Values of M and M' for the Google dataset with CPU resource

$h = 1$

	$M' = 1$	$M' = 5$	$M' = 12$	$M' = 100$
$M = 1$	0.055	0.068	0.071	0.106
$M = 5$	0.058	0.068	0.068	0.098
$M = 12$	0.059	0.048	0.046	0.050
$M = 100$	0.065	0.089	0.047	0.055

$h = 5$

	$M' = 1$	$M' = 5$	$M' = 12$	$M' = 100$
$M = 1$	0.088	0.073	0.076	0.108
$M = 5$	0.105	0.081	0.074	0.099
$M = 12$	0.117	0.079	0.076	0.097
$M = 100$	0.091	0.0899	0.078	0.101

$h = 10$

	$M' = 1$	$M' = 5$	$M' = 12$	$M' = 100$
$M = 1$	0.098	0.082	0.081	0.107
$M = 5$	0.121	0.095	0.080	0.099
$M = 12$	0.129	0.102	0.081	0.098
$M = 100$	0.104	0.112	0.084	0.101

In the remaining of this subsection, we use the sample-and-hold method (with $K = 3$) for forecasting and consider the impact of other aspects on the RMSE.

5.6.4.2 Different Clustering Methods

We consider the different clustering methods as in Section 5.6.3.2 combined with temporal forecasting. The RMSE results with different forecasting steps (h) are shown in Fig. 5.10. We see that our proposed approach performs the best in almost all cases. For long-term forecasting with large h, the static clustering method often performs similar as our proposed approach, because when there are fluctuations, dy-

namic clustering may not perform as good as static clustering for long time periods. Note, however, that the static clustering baseline is an offline method which requires knowledge of the entire time series beforehand, thus it is not really applicable in practice.

5.6.4.3 Different Values of M and M'

Table 5.3 shows the RMSE with different values of M and M' on the Google dataset with CPU resource, where we recall that M and M' are the number of time steps to look back into history when computing the similarity measure and forecasted cluster (and per-node offset), respectively (see Sections 5.5.2 and 5.5.3). We observe that the optimal choices of M and M' depend on the forecasting step h. Generally, $M = 1$ is a reasonably good value for all cases. The optimal value of M' tends to increase with h. This means that the farther-ahead we would like to forecast, the more we should look back into the history when determining the cluster membership and offset values of local nodes, which is intuitive because we need to rely more on long-term (stable) characteristics when forecasting farther ahead into the future. We choose $M' = 5$ as default in Section 5.6.1.2 which is a relatively good value for different h.

5.6.4.4 Proposed Similarity Measure vs. Jaccard Index

As discussed in Section 5.5.2, the Jaccard index used in [92] is another possible similarity measure that one could use. In Fig. 5.11, we compare the RMSE when using our proposed similarity measure and Jaccard index. Our proposed similarity measure gives a better or similar performance in all cases.

5.6.5 Comparison to Gaussian-based Method in [1]

Finally, we modify our setup and compare our proposed approach with the *Gaussian-based method* in [1].

The method in [1] includes separate training and testing phases, both set to 500 time steps (which is the value chosen in [1]). During the training phase, the central node receives measurements from every node (i.e., $B = 1$) and uses this information to select a subset of nodes ($K \ll N$) that will continue to send measurements during the testing phase. This subset of K nodes is called *monitors*. During the testing phase, the central node receives measurements only from the selected nodes (which is equivalent to having a transmission frequency of $B = \frac{K}{N}$), and the measurements of the non-monitor nodes are inferred based on the measurements from the monitors. There is no temporal forecasting in this mechanism.

We adapt our proposed approach to the above setting with separate training and testing phases as follows. During training, we perform K-means clustering, where we group nodes into clusters based on their 500 latest measurements (i.e., we perform K-means on 500-dimensional vectors). This gives us K clusters of nodes. We select one node in each cluster that has the smallest Euclidean distance from the centroid of this cluster. We consider this node as a monitor. During testing, we only receive measurements from the monitors. The resource utilizations at all nodes that belong to the same cluster as the monitor are estimated as equal to the measurement of the monitor. The minimum distance baseline in this setting is one that selects the K monitors randomly, and the other nodes are assigned to clusters based on their Euclidean distances from the monitors, where each cluster contains one monitor. Three algorithms that are proposed in [1] are also considered as baselines: Top-W, Top-W-Update, and Batch Selection, which are based on Gaussian models.

We only use 100 randomly selected machines in this experiment, because the approaches in [1] are too time-consuming to run on the entire dataset. The results

of RMSE defined on the estimation method described in this subsection above[8] are shown in Fig. 5.12 and the computational time of different approaches (on computer with Intel Core i7-6700 3.4 GHz CPU, 16 GB memory) is shown in Table 5.4. We see that our proposed approach provides the smallest RMSE, and it runs much faster than the three approaches (Top-W, Top-W-Update, and Batch Selection) from [1]. This observation is consistent with our discussions in Sections 5.2 and 5.3 that Gaussian models do not work well in our setting.

Table 5.4: Computation time (in seconds) for each approach and dataset (100 nodes)

	CPU Alibaba	**CPU Bitbrains**	**CPU Google**
Proposed	0.1401	0.16457	0.1370
Min.-distance	0.0231	0.0287	0.0238
Top-W	0.5987	0.6134	0.6074
Top-W-Update	29.3502	30.2132	27.4450
Batch Selection	2.8197	2.7812	2.2934

5.7 Conclusion

In this chapter, we have proposed a novel mechanism for the efficient collection and forecasting of resource utilization at different machines in large-scale distributed systems. The mechanism is a tight integration of algorithms for adaptive transmission, dynamic clustering, and temporal forecasting, with the goal of minimizing the RMSE of both spatial estimation and temporal forecasting. Extensive experiments on three real-world datasets show the effectiveness of our approach compared to baseline methods. Future work can study the integration of our approach with resource allocation, federated learning and other system management mechanisms.

[8]Note that this RMSE definition is different from that in earlier parts of this chapter.

Table 5.5: Summary of main notation for Chapter 5

Notation	Meaning
N	Number of nodes
K	Number of clusters (i.e. Number of different forecasting models)
d	Number of resource types (e.g., CPU, memory)
$\mathbf{x}_t = [x_{1,t}, ..., x_{N,t}]$	True measurements of N local nodes at time t (N tuples)
$x_{i,t}$	True measurement of node i, at time t, d-dimensional
$\mathbf{z}_t = [z_{1,t}, ..., z_{N,t}]$	Measurements stored at the central nodes of N local nodes at time t (N tuples)
$z_{i,t}$	Measurement stored at central node of node i, at time t, d-dimensional
$\beta_{i,t}$	Indication variable, $\beta_{i,t} = 1$ if node i sent most recent measurement at time step t, $\beta_{i,t} = 0$ otherwise
$C_{j,t}$	j-th cluster at time step t, (i.e. set of indices of local nodes whose measurements are included in the cluster)
$C_{j,t+h}$	Forecasted set of nodes in cluster j at time step $t + h$
$c_{j,t}$	Centroid of cluster j at time step t
$\hat{c}_{j,t+h}$	Forecast of centroid value at future time step $t + h$
h	Number of forecasting step
$\hat{s}_{i,t+h}$	Forecasted offset of node i with respect to the centroid of cluster j (to which node i is forecasted to belong to) at time step $t + h$
$\hat{x}_{i,t+h}$	Forecasted measurement of node i at time $t + h$
B_i	Maximum transmission frequency (for node i)
$F_{i,t}(\beta_{i,t})$	Penalty function that capture error of the measurement store at the central node for node i at time step t
$Q_i(t)$	Length of the "virtual queue" at node i
$w_{k,j}$	Similarity measure between cluster k and cluster i
M	Number of time steps to look back into the history when computing the intersection in the similarity measure
M'	Number of time steps to look back into the history when computing the offset
φ	One-to-one mapping from cluster k to cluster j

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

Federated learning allows machine learning algorithms to be trained on a broad range of data sets located at different locations. Specifically, federated learning enables multiple clients to collaborate on machine learning models without sharing their raw data. By allowing analytic over various data sources, federated learning has opened the door to many promising applications that involve sensitive data: from healthcare to financial services, from government use cases to consumer products of all kinds. In 2016, Google [2] formulated the first Federated learning algorithm, FedAvg, for solving federated learning problems, which however, faces some limitations when applied to real-world systems. These limitations are mainly related to expensive communications, spatial/temporal data heterogeneity, and systems heterogeneity.

This thesis proposes some enhancements to the FedAvg algorithm to improve resource efficiency and robustness of federated learning under heterogeneous and dynamic data. In Chapter 2, a new approach is proposed to optimize the FedAvg algorithm under resource constraints. The proposed approach learns the system and data characteristics in real-time and dynamically adapt the frequency of aggregation to maximize the learning accuracy for a given resource budget. Extensive experimentation results confirm the effectiveness of the proposed algorithm. Chapter 3 proposes a pre-processing step to the FedAvg algorithm to overcome noisy and irrelevant data at clients. More specifically, a method for selecting the subset of rele-

vant data to be involved in a federated learning task is proposed. The efficiency of the proposed data selection method is demonstrated through extensive experimental analysis using multiple real-world image datasets. In addition, Chapter 3 considers the co-existence of multiple tasks in federated learning systems and formulate a scheduling problem of these multiple tasks as a MILP, which is proved to be NP-hard, and proposed an efficient algorithm based on LP-relaxation and rounding to find its approximate solution.

With the goal in mind to address temporal data heterogeneity in federated learning, Chapter 4 presents a task-free continual learning approach that does not require storing training data, making it applicable to federated learning settings. Based on Bayesian Neural Networks (BNNs), the approach is able to continually learn on new data without forgetting much of what has been learned previously. The proposed approach also detects shifts in data, which allows the algorithm to work without knowing the task boundaries. Furthermore, it ensures that the global model remains within a maximum size so that the storage capacity for the saved models is not exceeded. The approach is validated on different continual learning scenarios. Finally, Chaper 5 presents a general mechanism that allows the central server to efficiently collects and forecasts the resource utilization at each client/device in a large-scale distributed system. Federated learning systems can further use the results provided by our mechanism for system management purposes.

6.2 Future Work

Some possible areas of future work are summarized as follows.

6.2.1 Adaptive Federated Learning with Heterogeneous Resources

In Chapter 2, we have proposed an algorithm able to dynamically adapt the frequency of aggregation to maximize the learning accuracy according to a given resource budget. This algorithm assumes *synchronous scenario*. In other words, the central server has to wait for model updates from all clients before performing aggregation. This means that the cost of global aggregation and local updates are defined by the slowest node. To avoid the whole federated learning process to be slow down by the slowest clients, *asynchronous scenario* can be further investigated. Asynchronous model updates would allow more efficient use of heterogeneous resources.

6.2.2 Exploring Potential of BNNs for Federated Learning

In Chapter 4, we have introduced a task-free continual learning approach based on BNNs which is applicable to federated learning. An interesting direction will be to integrate this Bayesian approach to a federated learning system and compare Bayesian-based federated learning with the traditional federated learning (i.e., FedAvg) for different scenarios (i.e., stationary data and non-stationary data distribution). In the Bayesian framework, each client can train its own set of BNNs and sends its selected set of BNN models to the central server. The central server would make predictions in similarly as described in section 4.4.2.

6.2.3 System Management for Federated Learning using Resource Utilization Forecasts

Chapter 5 focuses on the collection and forecasting of resource utilization and leave its application to system management for future work. To extend this line of work, one can use the resource's availability predictions provided by our approach to select an appropriate subset of clients to participate in the federated learning process.

Bibliography

[1] S. Silvestri, R. Urgaonkar, M. Zafer, and B. J. Ko, "An online method for minimizing network monitoring overhead," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 268–277. xi, xvi, 86, 88, 89, 91, 113, 116, 117

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016. 1, 2, 4, 10, 12, 14, 16, 25, 38, 119

[3] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of operations research*, vol. 1, no. 2, pp. 117–129, 1976. 8, 33, 48, 49

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. 11, 13, 23

[5] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881. 11, 12

[6] Y. Zhang, M. J. Wainwright, and J. C. Duchi, "Communication-efficient algorithms for statistical optimization," in *Advances in Neural Information Processing Systems*, 2012, pp. 1502–1510. 11, 12

[7] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," in *Advances in neural information processing systems*, 2015, pp. 1756–1764. 11

[8] C. Ma, J. Konečnỳ, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, "Distributed optimization with arbitrary local solvers," *Optimization Methods and Software*, vol. 32, no. 4, pp. 813–848, 2017. 12

[9] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2019. 12, 18, 31

[10] T. Tuor, S. Wang, K. K. Leung, and K. Chan, "Distributed machine learning in coalition environments: overview of techniques," in *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, 2018, pp. 814–821. 15

[11] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and trends in Machine Learning*, vol. 8, no. 3-4, 2015. 17

[12] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017. 19

[13] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. 23, 24

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 24, 50, 53

[15] S. Moro, P. Rita, and B. Vala, "Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach." *Journal of Business Research*, vol. 69, no. 9, pp. 3341–3351, 2016. 24

[16] H. Kahraman, S.Sagiroglu, and I.Colak, "Developing intuitive knowledge classifier and modeling of users' domain dependent data in web," *Knowledge Based Systems*, vol. 37, pp. 283–295, 2013. 24

[17] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *ICLR Workshop Track*, 2016. 25

[18] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019. 34, 38, 47, 52

[19] N. H. Tran, W. Bao, A. Zomaya, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1387–1395. 34

[20] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7. 34

[21] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434. 34

[22] L. Wang, W. Wang, and B. Li, "CMFL: Mitigating communication overhead for federated learning," in *IEEE ICDCS*, 2019. 34

[23] P. Blanchard, R. Guerraoui, J. Stainer *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 119–129. 34

[24] S. Shen, S. Tople, and P. Saxena, "A uror: defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 508–519. 34

[25] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, 2020, pp. 301–316. 34

[26] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520. 34

[27] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia, "Iterative learning with open-set noisy labels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8688–8696. 35, 51

[28] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," in *ICLR (Workshop)*, 2015. 35

[29] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *Workshop contribution at ICLR*, 2015. 35

[30] G. Patrini, A. Rozza, A. Krishna Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1944–1952. 35

[31] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *Advances in Neural Information Processing Systems*, 2018, pp. 8527–8537. 35

[32] A. Ghosh, H. Kumar, and P. Sastry, "Robust loss functions under label noise for deep neural networks," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 35

[33] Y. Li, J. Yang, Y. Song, L. Cao, J. Luo, and L.-J. Li, "Learning from noisy labels with distillation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1910–1918. 35

[34] A. Veit, N. Alldrin, G. Chechik, I. Krasin, A. Gupta, and S. Belongie, "Learning from noisy large-scale datasets with minimal supervision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 839–847. 35

[35] A. Vahdat, "Toward robustness against label noise in training deep discriminative neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 5596–5605. 35

[36] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 456–10 465. 35, 51

[37] X. Yu, B. Han, J. Yao, G. Niu, I. Tsang, and M. Sugiyama, "How does disagreement help generalization against label corruption?" in *International Conference on Machine Learning*, 2019, pp. 7164–7173. 35

[38] K. Lee, S. Yun, K. Lee, H. Lee, B. Li, and J. Shin, "Robust inference via generative classifiers for handling noisy labels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3763–3772. 35

[39] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951. 42

[40] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly*, vol. 1, no. 1, pp. 61–68, 1954. 46

[41] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011. 50, 53, 76

[42] S. Caldas, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019. 50, 53

[43] T. E. De Campos, B. R. Babu, M. Varma *et al.*, "Character recognition in natural images." *VISAPP (2)*, vol. 7, 2009. 50, 53

[44] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017. 50, 53

[45] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009. 50, 53

[46] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Con-*

ference on Computer and Communications Security, 2017, pp. 1175–1191. 64

[47] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016. 66, 67

[48] J. Wen, Y. Cao, and R. Huang, "Few-shot self reminder to overcome catastrophic forgetting," *arXiv preprint arXiv:1812.00543*, 2018. 66, 67

[49] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476. 66, 67, 76

[50] D. Feldman and M. Langberg, "A unified framework for approximating and clustering data," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 2011, pp. 569–578. 66, 67

[51] V. Braverman, D. Feldman, and H. Lang, "New frameworks for offline and streaming coreset constructions," *arXiv preprint arXiv:1612.00889*, 2016. 66, 67

[52] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010. 66, 67

[53] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 2990–2999. 66, 67, 76

[54] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying for-

getting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 66, 68

[55] X. He and H. Jaeger, "Overcoming catastrophic interference using conceptor-aided backpropagation," 2018. 67

[56] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems*, 2017, pp. 4652–4662. 67

[57] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017. 67, 81

[58] Y. Li, Z. Li, L. Ding, P. Yang, Y. Hu, W. Chen, and X. Gao, "Supportnet: solving catastrophic forgetting in class incremental learning with support data," *arXiv preprint arXiv:1806.02942*, 2018. 67

[59] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning," in *International Conference on Learning Representations*, 2018. 67

[60] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning in bayesian neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 75–78. 67, 68, 76

[61] R. Aljundi, K. Kelchtermans, and T. Tuytelaars, "Task-free continual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 254–11 263. 67

[62] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 816–11 825. 67

[63] A. Chrysakis and M.-F. Moens, "Online continual learning from imbalanced data," *Proceedings of Machine Learning Research*, 2020. 67

[64] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," *arXiv preprint arXiv:1902.10486*, 2019. 67, 77

[65] D. Rao, F. Visin, A. Rusu, R. Pascanu, Y. W. Teh, and R. Hadsell, "Continual unsupervised representation learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 7647–7657. 67

[66] S. Lee, J. Ha, D. Zhang, and G. Kim, "A neural dirichlet process mixture model for task-free continual learning," in *International Conference on Learning Representations*, 2019. 67, 77, 78

[67] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1613–1622. 69

[68] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," in *International Conference on Artificial Intelligence and Statistics*, 2020, pp. 3762–3773. 76, 81

[69] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013. 76

[70] K. Shridhar, F. Laumann, and M. Liwicki, "A comprehensive guide to bayesian convolutional neural network with variational inference," *arXiv preprint arXiv:1901.02731*, 2019. 77, 78

[71] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," in *International Conference on Learning Representations*, 2018. 81

[72] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*. IEEE, 2019, pp. 1–5. 85

[73] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7. 86

[74] B. Cai, R. Zhang, L. Zhao, and K. Li, "Less provisioning: A fine-grained resource scaling engine for long-running services with tail latency guarantees," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 30:1–30:11. [Online]. Available: http://doi.acm.org/10.1145/3225058.3225113 88

[75] M. Grechanik, Q. Luo, D. Poshyvanyk, and A. Porter, "Enhancing rules for cloud resource provisioning via learned software performance models," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '16. New York, NY, USA: ACM, 2016, pp. 209–214. [Online]. Available: http://doi.acm.org/10.1145/2851553.2851568 88

[76] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 35–45. [Online]. Available: http://dl.acm.org/citation.cfm?id=2821357.2821365 88

[77] H. Shen and L. Chen, "Resource demand misalignment: An important factor to consider for reducing resource over-provisioning in cloud datacenters," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1207–1221, June 2018. 88, 90

[78] G. Coluccia, E. Magli, A. Roumy, and V. Toto-Zarasoa, "Lossy compression of distributed sparse sources: a practical scheme," in *Signal Processing Conference, 2011 19th European*. IEEE, 2011, pp. 422–426. 88, 108

[79] J. E. Barceló-Lladó, A. M. Pérez, and G. Seco-Granados, "Enhanced correlation estimators for distributed source coding in large wireless sensor networks," *IEEE Sensors Journal*, vol. 12, no. 9, pp. 2799–2806, 2012. 88, 108

[80] C. Anagnostopoulos and S. Hadjiefthymiades, "Advanced principal component-based compression schemes for wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 1, p. 7, 2014. 88, 108

[81] Y. Li and Y. Liang, "Compressed sensing in multi-hop large-scale wireless sensor networks based on routing topology tomography," *IEEE Access*, vol. 6, pp. 27 637–27 650, 2018. 88, 108

[82] M. Leinonen, M. Codreanu, and M. Juntti, "Compressed acquisition and progressive reconstruction of multi-dimensional correlated data in wireless sen-

sor networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6449–6453. 88, 108

[83] Y. Zhang, T. N. Hoang, K. H. Low, and M. S. Kankanhalli, "Near-optimal active learning of multi-output gaussian processes." in *AAAI*, 2016, pp. 2351–2357. 88, 89, 91

[84] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 235–284, 2008. 88, 89, 91

[85] C. Liu, K. Wu, and M. Tsao, "Energy efficient information collection with the arima model in wireless sensor networks," in *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, vol. 5. IEEE, 2005, pp. 5–pp. 88

[86] Y. W. Law, S. Chatterjea, J. Jin, T. Hanselmann, and M. Palaniswami, "Energy-efficient data acquisition by adaptive sampling for wireless sensor networks," in *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*. ACM, 2009, pp. 1146–1151. 88

[87] H. Harb, A. Makhoul, A. Jaber, R. Tawil, and O. Bazzi, "Adaptive data collection approach based on sets similarity function for saving energy in periodic sensor networks," *International Journal of Information Technology and Management*, vol. 15, no. 4, pp. 346–363, 2016. 88

[88] S. Chatterjea and P. Havinga, "An adaptive and autonomous sensor sampling frequency control scheme for energy-efficient data acquisition in wireless sensor networks," in *International Conference on Distributed Computing in Sensor Systems*. Springer, 2008, pp. 60–78. 88

[89] A. K. Idrees and A. K. M. Al-Qurabat, "Distributed adaptive data collection protocol for improving lifetime in periodic sensor networks." *IAENG International Journal of Computer Science*, vol. 44, no. 3, 2017. 88

[90] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 554–560. 89

[91] K. S. Xu, M. Kliger, and A. O. Hero III, "Adaptive evolutionary clustering," *Data Mining and Knowledge Discovery*, vol. 28, no. 2, pp. 304–336, 2014. 89

[92] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *Advances in social networks analysis and mining (ASONAM), 2010 international conference on*. IEEE, 2010, pp. 176–183. 89, 99, 115

[93] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin, "Detecting communities and their evolutions in dynamic social networks—a bayesian approach," *Machine learning*, vol. 82, no. 2, pp. 157–189, 2011. 89

[94] P. C. Ma, K. C. Chan, X. Yao, and D. K. Chiu, "An evolutionary clustering algorithm for gene expression microarray data analysis," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 296–314, 2006. 89

[95] C. Rana and S. K. Jain, "An evolutionary clustering algorithm based on temporal features for dynamic recommender systems," *Swarm and Evolutionary Computation*, vol. 14, pp. 21–30, 2014. 89

[96] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux, "Intel lab data," *Online dataset*, 2004. 90

[97] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011. 90, 101

[98] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, May 2009. 94

[99] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010. 96

[100] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979. 97

[101] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. 99

[102] T. W. Liao, "Clustering of time series data—a survey," *Pattern Recognition*, vol. 38, no. 11, pp. 1857 – 1874, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320305001305 99

[103] G. E. Box, G. M. Jenkins, and J. F. MacGregor, "Some recent advances in forecasting and control," *Applied Statistics*, pp. 158–179, 1974. 100

[104] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 100

[105] "Alibaba trace," https://github.com/alibaba/clusterdata/tree/v2018, 2018. 101

[106] S. Shen, V. Van Beek, and A. Iosup, "Workload characterization of cloud datacenter of bitbrains," *TU Delft, Tech. Rep. PDS-2014-001*, 2014. 101

[107] K. P. Burnham and D. R. Anderson, *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Science & Business Media, 2003. 103