SENSOR ANALYTICS USING MULTI-TASK LEARNING AND FEDERATED LEARNING

by

Yujing Chen

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Huzefa Rangwala, Dissertation Director
_____	Dr. Hakan Aydin, Committee Member
_____	Dr. David Rosenblum, Committee Member
_____	Dr. Dov Gordon, Committee Member
_____	Dr. Martin Slawski, Committee Member
_____	Dr. David Rosenblum, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Summer 2021
George Mason University
Fairfax, VA

Sensor Analytics Using Multi-Task Learning and Federated Learning

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Yujing Chen
Master of Art
Renmin University of China, 2016
Bachelor of Engineering
Beijing University of Posts and Telecommunications, 2010

Director: Dr. Huzefa Rangwala, Professor
Department of Computer Science

Summer 2021
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my husband who has offered unwavering support and encouragement during my doctoral journey. This dissertation is also dedicated to my parents who have always been a great source of support.

Acknowledgments

I would like to thank my supervisor, Dr. Huzefa Rangwala for his invaluable advice, continuous support, and patience during my PhD study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life.

I would also like to extend my thanks to the committee membbers, Dr. David Rosenblum, Dr. Hakan Aydin, Dr. Dov Gordon and Dr. Martin Slawski for their technical support and suggestion on my dissertation work. I am also thankful to my friends, lab mates for a cherished time spent together in the lab, and in social settings.

Finally, my appreciation also goes out to my family for their encouragement and support all through my studies.

Table of Contents

List of Tables

List of Figures

Abstract

SENSOR ANALYTICS USING MULTI-TASK LEARNING AND FEDERATED LEARNING

Yujing Chen, PhD

George Mason University, 2021

Dissertation Director: Dr. Huzefa Rangwala

Sensors and internet of things (IoTs) are ubiquitous in our modern day-to-day living. The past decade has been marked by the rapid emergence and proliferation of a myriad of small devices. Applications range from smart home devices that control cooking ranges to mobile phones, wearable devices that serve as fitness trackers and personalized coaches. There is a critical need for the analysis of heterogeneous multivariate temporal data obtained from individual sensors.

Multi-task learning (MTL) is a machine learning framework which learns a shared representations across related tasks and naturally suited for sensor data learning. I first propose two approaches by utilizing multi-task learning framework with attention mechanism, which can jointly trains classification/regression models from multiple related tasks where data on each task is generated from one or more sensors. The temporal and non-linear relationships underlying the captured data are modeled using convolution neural network (CNN) and long-short term memory (LSTM) models. And the attention mechanism seeks to learn shared feature representations across multiple tasks for improving the overall generalizability of the machine learning model.

However, as massive data is generated from the edge devices, combined with the growth

of computing power on these devices, it is attractive to learn models directly over networks with data on these distributed devices. Federated learning (FL) is a machine learning paradigm where a shared central model is learned across distributed edge devices while the training data remains on these devices. Federated Averaging (*FedAvg*) is the leading optimization method for training non-convex models in this setting with a synchronized protocol. But the assumptions made by FedAvg are not realistic given the heterogeneity of devices. In particular, the volume and distribution of collected data vary in the training process due to different sampling rates of edge devices and users' preference change. The edge devices themselves also vary in their available communication bandwidth and system configurations, such as memory, processor speed, and power requirements. This leads to vastly different training times as well as model/data transfer times. Furthermore, availability issues at edge devices can lead to a lack of contribution from specific edge devices to the federated model. Therefore, I further propose an asynchronous online approach and another collaborative work with tiers in the federated learning framework to tackle these issues. Specially, to deal with the distribution change problem of the streaming data, I propose one more approach for online federated learning with concept drift. In this work, I apply techniques on both local learning and the central model aggregation to alleviate the effect on the global model performance with local data concept drift. In addition, a new method is proposed to select local clients at each round to reduce the overall communication costs.

Motivated by the growing interest and challenges in federated learning research, this thesis demonstrates the strengths of academic performance on both prediction and communication-efficient performance on multiple benchmarks. Incorporating the techniques in this thesis to develop federated learning systems with more accurate prediction, lower communication costs and resource-efficient learning. More importantly, effective online approaches are proposed to tackle the issues with streaming data in real-life setting.

Chapter 1: Introduction

1.1 Motivation

Sensors inter-connected via the internet also referred by Internet of Things (IoTs) are everywhere; on us, in our homes, and our living environment. They seek to improve the quality of our daily life. For example, wearable devices provide consumer-centric healthcare solutions and move towards the ability of providing technological innovations, where the health information of a person can be seamlessly integrated into various everyday activities. Besides the wearable devices, sensors in other forms like smartphones, infrastructure components (cameras, weather recorders, etc,.) and other low-cost sensors (e.g., WiFi) are indispensable parts in many domains. Inexpensive hardware, widespread communication networks, decreased computational infrastructure costs and heavy commercial investment have lead to the following projections: (i) 40% year-over-year growth in overall data captured by these sensors [1] and (ii) 75 billion IoT devices by 2025 [2]. With the advancement of machine learning and data mining approaches, together with these sensing and detection capabilities, feasible and appealing end-to-end solutions are possible in many domains (healthcare, activities of mobile phone users, environment monitoring, etc,.)

Developing effective machine learning methods for sensor data study is important and needed for several reasons. First, these ubiqiutious sensors produce massive volumes of data in a given day. Many of these sensors may have communication challenges in terms of bandwidth; power challenges as they may be low-powered and computational challenges in terms of their systems performance. Therefore, distributed model training over a large number of edge nodes is essential for generating useful knowledge. Second, the study on these sensor data allows for fine-grained sensing and inference of users' context, physiological signals, and even mental health states. The sensing and detection capabilities coupled with

advanced data analytics are leading to intelligent intervention and persuasion techniques. Third, with the growth in computation power of these edge devices, it is possible to build viable solutions to push the training of statistical models to the edge. Data is stored on device to protect user privacy in certain aspects [3–5]. It may not be suitable to share the data obtained from these sensors with a cloud-based server or a central server. Fourth, data on devices may increase during training, since sensors on these distributed devices usually have a high sampling frequency. Additionally, the underlying distribution of the data can change arbitrarily over time (concept drift) as the generation of data streams is usually in the non-stationary environment. Thus effective strategies should be incorporated to deal with the variation of the streaming device data.

1.2 Research Questions

The central research questions that this study aims to answer is, *How to develop effective and high performance approaches for sensor data learning?* This thesis will address the following sub-questions:

1. How to model the relationships among related sensor devices and train accurate and generalizable machine learning models ?

2. How to develop distributed networks with on-device learning?

3. Given the heterogeneity across devices in terms of availability, how to design adaptable learning algorithms ?

4. How to to reduce the communication cost in transferring model parameters among server and devices?

5. If the distribution of local data changes during the learning process, how to alleviate this to ensure robust local and global machine learning models.?

1.3 Problems, Challenges and Contributions

Sensors produce massive heterogeneous sequential data. Take the activities data of mobile phone users as an example, to better utilize the data from sensors, effective predictive models for each user are expected. However, due to reasons such as power (battery level) and storage, device heterogeneity, the amount and quality of the data for certain user are not sufficient to learn a robust machine learning model.

In this thesis I first solve this problem through a *multi-task learning* (MTL) framework, where the goal is to improve generalization performance by transferring knowledge among multiple related models.

MTL is a machine learning framework which learns a shared representations across related tasks [6]. The goal of MTL is to improve the overall prediction quality by learning the tasks jointly and sharing information between them. I refer each independent model learned with data from multi-modal sensors as a task in this work. Specifically, I seek to learn a regression/classification model for each task. MTL approach simultaneously train multiple related tasks and have found success in producing generalizable machine learning models for various domains and applications [7–10].

Apart from learning a shared generative framework, most data generated by these sensors are multivariate temporal sequences and are typically stored in distributed devices. These kinds of data cannot be transferred over networks to a central server (cloud) for many reasons: 1) data may be sensitive; containing personal information and 2) network bandwidth may prevent uploading large volumes data to a data center. Further, the high dimensionality, high feature correlation, and the (typically) large amount of noise that characterize the collected data present difficult challenges.

In my first MTL work, I model the sequential data from sensors using recurrent neural networks (RNNs) [11] adapted for handling long-term dependencies; commonly referred to as long short-term memory models (LSTM) [12]. These sequential deep learning models

have found success in applications including image captioning and segmentation [13], language modeling [14] and speech recognition [15]. Attention based models have shown success at extracting an effective feature representation (e,g,. [16, 17]). In order to capture a better feature representation among tasks, thereby learn effective models for each task, I apply the popular attention mechanism on the shared feature information. I present a novel multi-task deep learning approach, M-Att, which uses attention mechanism to better learn a common feature representation among multiple tasks and show promising results on several public datasets.

In the second MTL work, I use a federated learning setting to learn a model across distributed related tasks. In particular, I address the aforementioned challenges within the framework of federated multi-task learning (MTL). This allows for multiple benefits: (i) reduced communication costs by not sending large volumes of data to a central server; (ii) distributed learning and (iii) privacy guarantees when the data resides on only the device. Similar to prior work on federated multi-task learning [18], data associated with each task stays local. Individual models are learned for each task first, and a central node controls the communication between local model parameters and global shared parameters.

The above mentioned two methods are within the multi-task learning frameworks. Multi-task learning models are not suitable for on device training given that they assume all clients (devices) participate in each training round. This requires that all clients are available because each client is training an individual specific model [19]. However, edge devices could be frequently offline during the training process due to unreliable networks or other factors. With growth in popularity and computation power of these edge devices, federated learning (FL) has emerged as a potentially viable solution to push the training of statistical models to the edge [3–5]. As shown in Figure 1.1, FL involves training a shared global model from a federation of distributed devices under the coordination of a central server; while the training data is kept on the edge device. Each edge device performs training on its local data and updates model parameters to the server for aggregation. Many applications can leverage this FL framework such as learning activities of mobile device

users, forecasting weather pollutants and predicting health events (e.g., heart rate).

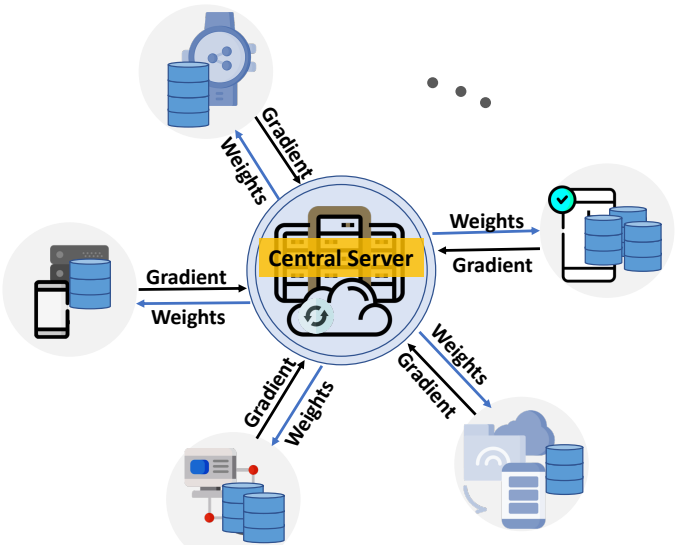


Figure 1.1: Illustration of Federated Learning network. Local devices work together to learn a shared global model with the collaboration of a Central server.

Many prior FL approaches use a synchronous protocol (e.g., *FedAvg* [3] and its extensions [4,5,20–22]), where at each global iteration, the server distributes the central model to a selected portion of clients and aggregates by applying weighted averaging after receiving all updates from these clients. These methods are costly due to a synchronization [23] step (shown in Figure 1.2), where the server needs to wait for all local updates before aggregation. The existence of lagging devices (i.e., stragglers, stale workers) is inevitable due to device heterogeneity and unreliability in network connections. To address this problem, asynchronous federated learning methods [24, 25] were proposed, where the server can aggregate without waiting for the lagging devices. However, these asynchronous frameworks assume a fixed magnitude of device data during the training process, which is not practical in real-life settings. Data on local devices may increase during training, since sensors on these distributed devices usually have a high sampling frequency.

To summarize, there are several challenges that cannot be handled by the aforementioned

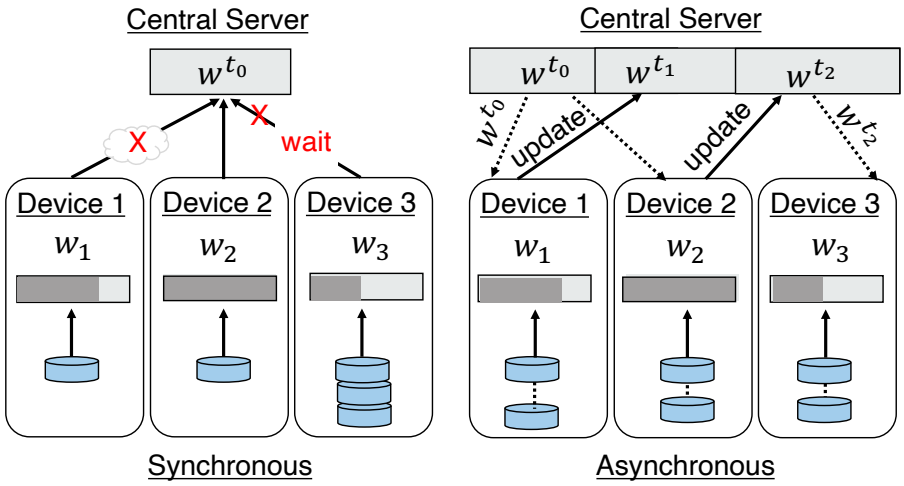


Figure 1.2: Illustration of Synchronous vs. Asynchronous update. In synchronous optimization, Device 1 has no network connection and Device 3 needs more computation time, thus the central server has to wait. Asynchronous updates do not need to wait.

synchronous or asynchronous federated optimization approaches: 1) mobile devices can be frequently offline or have poor communication bandwidth due to network constraints. Consequently, the synchronized federated learning frameworks can be extremely slow; 2) edge devices may lag or even dropout due to data or system heterogeneity [18]; 3) data on local devices may increase and vary with regards to the data-generating distribution during the training process, therefore the inter-client relatedness could potentially vary over time; 4) the non-IID[1] (not independent and identically distributed) and highly imbalanced characteristics of device data lead to challenges in effective model training [18].

To address the challenges outlined above, I propose an asynchronous online federated learning framework (ASO-Fed), where the central model does not wait for collecting and aggregating the gradient information from lagging clients, and clients perform online learning to deal with local streaming data. The proposed ASO-Fed approach is shown to improve prediction performance and converge fast with lower computation cost on real-world and

[1]A detailed definition can be found in Chapter 2.2.

benchmark datasets. To the best of our knowledge, this is one of the first studies on asynchronous federated learning with local streaming data.

In addition, I also present a Federated learning method with Asynchronous Tiers (FedAT) under non-IID data. This work is a collaborative work and aims to tackle the straggler and communication bottleneck issues in federated learning.

ASO-Fed provides an efficient solution for the problem of continuous arriving data during the FL training process. However, what if the underlying distribution of certain device data changes dramatically? Most of these device data are data streams, which are produced continuously by the device sensors [26]. In general, most of the existing FL solutions are under the hypothesis that these device data are stationary. However, in the real-world, the underlying distribution of the data can change arbitrarily over time as the generation of data streams is usually in the non-stationary environment. This phenomenon is known as concept drift [27, 28], which exists commonly in the scenarios of large scale data learning. For example, user activities prediction models trained before the COVID-19 may not work equally well during COVID-19 pandemic as the user's behaviors changes, weather prediction models change from season to season, customer consumption patterns and underlying recommender systems vary due to the pandemic, economy and so on. In other research fields, concept drift has also been defined using alternative names such as covariate shift [29], dataset shift [30] and non-stationary learning [31].

In federated learning process, the occurrence of concept drift of local device data can lead to a tremendous drop on prediction accuracy. Therefore, it is crucial that the federated learning algorithms work adaptively. Finally, to deal with the distribution change situation of local device data, I propose a novel asynchronous federated framework with concept drift learning which can alleviate the effect on the performance of the central model, and reduce the overall communication cost compared with ASO-Fed.

The main contributions of my work are as follows:

- I propose M-Att with a new perspective of information sharing scheme for multi-task learning, which using attention mechanism in multi-task deep learning framework to

learn a shared representation among related tasks [32].

- I model and incorporate LSTM with a hierarchical attention mechanism to evaluate personal feature correlations for each local device and learn a cross-device temporal correlations among devices. This proposed approach (FATHOM) can learn separate models for each sensor device and keep data locally[33].

- I design an asynchronous federated learning framework (ASO-Fed) under a non-IID setting that allows updates from clients with continuously arriving data. The proposed framework learns inter-client relatedness effectively using regularization and a central feature learning module. Experiments on multiple benchmarks show that ASO-Fed is robust against data heterogeneity and network connections with high communication delays between the server and some clients. I provide theoretical guarantees for the convergence of this proposed model[34].

- To deal with the concept drift issue on local device data during the training process, propose an asynchronous federated learning framework, FedConD, to detect and handle the sudden data distribution changes of local devices. Additionally, the server adopts a novel communication strategy to speedup the model convergence and balance the frequency of local updates.

- A co-authored work of tiered federated learning framework that updates local model parameters synchronously within tiers and update the global model asynchronously across tiers has been developed to deal with the straggler and communication-bottleneck problems [35].

Chapter 2: Background

2.1 Preliminaries and Notations

Given, K distributed devices (tasks)[1] with inputs $\{(\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}), ..., (\mathbf{X}^{(2)}, \mathbf{Y}^{(k)}), ..., (\mathbf{X}^{(K)}, \mathbf{Y}^{(K)})\}$. Each device consists of D data series collected from multiple sensors. Let $\mathcal{D}_k$ denote data captured on device k, and define $n_k = |\mathcal{D}_k|$. For each device k, given a time window of length T, we use $\mathbf{X}^{(k)} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\} \in R^{D \times T}$ to denote the readings of all features, and $\mathbf{Y}^{(k)} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_T\} \in R^{M \times T}$ to represent the target series during T window length, where $M (M \geq 1)$ is the number of target label columns. We denote $N = \sum_{k=1}^{K} |\mathcal{D}_k|$ as the total number of samples in K devices. Assuming for any $k \neq k^{'}$, $\mathcal{D}_k \bigcap \mathcal{D}_{k'} = \varnothing$.

With labeling function $\{f(\mathbf{X}) \rightarrow \mathbf{Y}\}$, the goal is to learn a model for a hypothesis set $\mathcal{S} \subset \{s : \mathbf{X} \rightarrow \mathbf{Y}\}$ using the training data and minimizing the average error across all the K devices:

$$\min \Big(\frac{1}{K} \sum_{k=1}^{K} \ell(s(\mathbf{X}^{(k)}), f(\mathbf{X}^{(k)})) \Big) \tag{2.1}$$

For multi-label classification tasks we train the networks to minimize the cross-entropy of the predicted and true distributions for each device k with N instances across the M labels.

$$\ell(\hat{\mathbf{Y}}, \mathbf{Y}) = -\frac{1}{MN} \sum_{m=1}^{M} \sum_{i=1}^{N} \hat{y}_i^m log(y_i^m) \tag{2.2}$$

[1] In the multi-task work, individual model of a user or a location with multiple sensors is considered as a task.

Table 2.1: Important Notations.

Notation	Meaning
D	# of features in each device
M	# of labels in each device
n_k	# of instance in each device
N	# of instance in all devices
K	# of devices
T	size of time window
$\mathbf{X}^{(k)}$	input features of device k
$\mathbf{X}_T^{(k)}$	features of device k with window size T
$\mathbf{Y}^{(k)}$	target label matrix of device k
$\hat{\mathbf{Y}}_T^{(k)}$	predicted label matrix of device k with window size T
a_t	attention weight
$\mathbf{c}_t^{(k)}$	attention vector
$\mathbf{a}_d^{(k)}$, $\mathbf{a}_t$	attention weight of task-specific level and global temporal level, respectively
$\boldsymbol{\phi}_d^{(k)}$	task-specific level context vector
$\boldsymbol{\psi}_t^{(k)}$	temporal context vector of device k

where $\ell(\hat{\mathbf{Y}}, \mathbf{Y})$ is the loss function. $\hat{\mathbf{Y}}$ denotes the predicted labels and $\mathbf{Y}$ is the true labels.

For multi-output regression tasks, we train the networks to minimize the mean absolute error of the predicted and true distributions for each device k with N instances across the M outputs.

$$\ell(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{MN} \sum_{m=1}^{M} \sum_{i=1}^{N} |\hat{y}_i^m - y_i^m| \tag{2.3}$$

We provide the primary notations of this dissertation in Table 2.1.

2.2 IID versus non-IID

Following Li et al. [36], we have the following definitions for IID and non-IID.

Definition 1. (*Independent and identically distributed (IID)*). Suppose the data in the $k-$th device are i.i.d. sampled from the distribution $\mathcal{D}_k$. Then the overall distribution is a mixture of all local data distributions: $\mathcal{D} = \sum_{k=1}^{K} \frac{n_k}{N} \mathcal{D}_k$, n_k is the number of data samples on device k and N is the number of total samples across all devices. The prior work assumes the data are iid generated by or partitioned among the K devices [36–40], that is, $\mathcal{D}_k = \mathcal{D}$ for all $k \in [K]$. However, real-world applications do not typically satisfy the IID assumption.

Definition 2. (*Not Independent and identically distributed (non-IID)*). Suppose the data in the $k-$th device are sampled from the distribution $\mathcal{D}_k$, and the overall distribution is a mixture of all local data distributions: $\mathcal{D} = \sum_{k=1}^{K} \frac{n_k}{N} \mathcal{D}_k$. We have $\mathcal{D}_k \neq \mathcal{D}$ for all $k \in [K]$, that is, a device's local data cannot be regarded as samples drawn from the overall distribution. The data available locally fail to represent the overall distribution. Furthermore, the data distributions in the $k-$th and $l-$th devices, denote $\mathcal{D}_k$ and $\mathcal{D}_l$, can be different.

Device data for federated learning usually follows the non-IID distribution, due to data is generated from devices in different places (e.g., device owned by different users, different sensors in a soil, weather sensors in different locations) and the non-stationary data generating environment i.e., data distributions can change over time.

2.3 Related Work on Multi-task Learning

Multi-task learning (MTL) is designed for the simultaneous training of multiple related tasks, which have the same prediction targets. Leveraging common information across related tasks has shown to be effective in improving the generalization performance of each task [6, 41–43]. MTL is particularly useful when there are a number of related tasks but some tasks have scarce amounts of available training data. Many researchers have investigated MTL from varied perspectives [8, 44]. Task relationships are modeled by sharing

layers/units of neural networks [6], sharing subset of features [45–48], sharing a low dimensional common subspace [49,50], or assuming a clustering among tasks [51,52], or achieving shared representation by structured regularization [53,54]. In this work we do not make any assumptions on task relationships beforehand and learn the common representation with attention mechanisms directly from the data.

Specifically, with the context of streaming data from multiple sensors, a multi-task multilayer perceptron (MLP) model [55] was developed to recognize different human activities. By manipulating the MLP model to fit uncontrolled in-the-wild unbalanced data, this approach outperformed a standard logistic regression (LR) model [56]. The MLP and LR approaches do not leverage the underlying temporal dependencies and inter-feature correlations of sensors' data. In this thesis I propose FATHOM within the federated setting (discussed in Section 4) by feeding the learned common feature representation across all tasks back to each local task, we get a much better performance of all tasks.

2.4 Related Work on Attention-based Multi-task Learning with Deep Network

Attention mechanisms have recently attracted enormous interest due to their highly parallel computation, significantly less training time, and flexibility in modeling dependencies. In multi-task learning, attention mechanisms can capture a better feature representation among tasks, thereby learn effective models for each task. Currently, most attention-based frameworks are using deep networks [57–59]. Fundamentally, neural networks allocate importance to input features through the weights of the learned model. In the context of deep learning, an attention mechanism allows a network to assign different levels of importance to various inputs by adjusting the weights [16]. This leads to a better feature representation.

Attention approaches can be roughly divided into Global Attention and Local Attention [57]. The global attention is akin to soft attention [58–60]; where the alignment weights are learned and placed "softly" over all patches in the source data. Local attention only

selects one patch of the data to access at a time. A special case of attention mechanism is self-attention (or intra-attention), which using different positions within a single sequence to compute a representation of this sequence [61]. Shen *et al.* [62] proposed a self-attention network for language understanding without any RNN/CNN structure.

Attention-based encoder-decoder neural network models have recently shown promising results in machine translation and speech recognition. Attention has been introduced in alignment-based RNN models [63, 64] for speech recognition and machine translation [57]. Liu *et al.* [63] introduced attention to alignment-based RNN model for speech recognition. Chorowski *et al.* [64] used the attention mechanism in RNN with alignment technique for speech recognition. Luong *et al.* [57] proposed using attention mechanism for machine translation. Yang *et al.* [65] proposed hierarchical attention networks which has two levels of attention mechanisms applied at the word and sentence-level for document classification. Liang *et al.* [66] provided a multi-level attention-based recurrent neural network for modeling spatio-temporal sensor data.

Our proposed M-Att (discussed in Section 3)and FATHOM utilize attention mechanisms to learn better feature representation across all devices, especially, FATHOM with hierarchical attention learns individual feature correlations within each local task. It also learns shared feature representations across distributed tasks with a central attention mechanism that focuses on time steps. By passing this central time attention back to each local task, we are able to learn task-specific representations.

2.5 Related Work on Distributed Optimization

Within the field of large-scale distributed machine learning, many optimization methods have been developed in the past few decades. One commonly used traditional distributed optimization approach is alternating direction method of multipliers (ADMM) [67]. Even though it can be used in a large-scale distributed optimization setting, the introduction of local copies increases the number of iterations to achieve good performance. As massive

data is generated from edge devices such as mobile phones, wearable devices and sensors, combined with the growth of computing power on these devices, it is attractive to learn models directly over networks with data on these distributed devices. Multi-task learning models are not suitable for edge device training; since they assume that all clients (devices) participate in each training round. This requires that all clients are available because each client is training an individual specific model [19]. However, edge devices could be frequently offline during the training process due to unreliable networks or other factors. The parameter server approaches [68, 69], where server nodes maintain globally shared parameters and data distributed on local nodes, however, often suffer from problems such as high network bandwidth or communication overhead [3]. Numerous other distributed optimization methods [69–74] in recent years are also not suitable for on-device learning. Federated Learning provides a promising solution that is capable of dealing with heterogeneous devices across adhoc communication networks.

2.5.1 Federated learning

As mentioned in the above paragraph, federated learning has emerged as a potentially viable solution to enable the training of statistical models locally on the devices [3, 5]. Federated learning seeks to train a predictive model while training data is distributed on distributed client nodes. Federated learning has been proposed as an alternative setting for decentralized approaches in [3]: leave training data distributed on local devices, and a shared central model is learned by aggregating locally-computed updates. Compared with conventional distributed machine learning [75–78], this framework is more robust to highly unbalanced non-i.i.d data, unstable network connections, and large number of clients. Konečnỳ *et al.* [5, 79] proposed methods to reduce the communication cost in federated learning and a new setting to deal with the federated optimization problem. The above research on federated learning aims at learning a single model across the network. Different from these works, Smith *et al.* [18] provided an approach to solve statistical challenges in the federated setting of multi-task learning. In particular, Smith *et al.* [18] solve issues

related to high communication costs, stragglers (nodes with less computation power), and reliability. We adopt this federated multi-task learning setting. However, our method is different from Smith *et al.* [18], as we focus on improving model performance across all local tasks using a hierarchical attention mechanism.

Federated optimization methods have shown significant improvements on balancing communication versus computation over traditional distributed approaches [67, 80]. Federated learning was first introduced by McMahan *et al.* [3] and has been benchmarked on image and language datasets. Many extensions have been explored based on this original federated learning setting [4, 5, 20, 81, 82]. A better approach to deal with non-IID data distribution is proposed by sharing a small amount of data with other devices [83]. However, all these studies update the federated model in a synchronous fashion and do not tackle the problem of stragglers and dropouts.

Smith *et al.* [18] developed a primal-dual optimization method within a multi-task learning paradigm. This involved learning separate models for each device and dealing with stragglers. However, this approach was not suitable for non-convex formulations (e.g., deep learning), where strong duality is no longer guaranteed. Xie *et al.* [24] proposed an asynchronous update procedure for federated optimization by updating the global model with weighted averaging, but this did not consider real-world scenarios where edge devices faced continuous streaming data. Our proposed ASO-Fed assumes no constraints on the server aggregation procedure and provides the performance guarantees on clients with heterogeneous data. We also incorporate online learning on clients to leverage the continuous arrival of new data points. Additionally, ASO-Fed can converge well with randomly inactive devices (e.g., stragglers, dropouts), we provide convergence analysis on model convergence without the consideration of device inactivity and the data heterogeneity. In recent work, Ruan et al. [84] provide a thorough analysis on how flexible device participation can affect the learning convergence.

2.5.2 Communication-Efficient Federated Optimization

McMahan et al. propose a FL approach called FedAvg [3], where instead of communicating after every iteration, each client performs multiple iterations of SGD to compute a weight update. By reducing the communication frequency, FedAvg reduces the communication cost and can work with partial client participation. In a follow-up work, Konečnỳ et al. [5] propose two approaches to reduce the uplink communication costs, i.e., structured updates and sketched updates, combined with probabilistic quantization. These two approaches, however, are only suitable for i.i.d. settings in FL. For Non-i.i.d. settings, they can significantly slow down the convergence speed in terms of SGD iterations.

In the broader realm of communication-efficient distributed deep learning, a wide variety of methods has been proposed to reduce the amount of communication during the training process. Chen et al. [85] propose a layerwise asynchronous update scheme that updates the parameters of the deep layers less frequently than those of the shallow layers. Mills et al. [86] adapt FedAvg to use a distributed form of Adam optimization and compress the uploaded parameters. Jeong et al. [87] develop federated distillation that follows an online version of knowledge distillation to compress the model. Reisizadeh et al. [88] present a periodic averaging and quantization approach to reduce communication costs. However, these works only target uplink communication compression and are developed for synchronous frameworks that neglect the real-world scenario where stragglers are common. In addition to the above mentioned server-client topology, solving communication bottlenecks via quantization and compression has also gained considerable attention in decentralized training [89–91]. While such techniques can be used to reduce communication costs in FL, a decentralized network topology in distributed learning without a server is fundamentally different and is thus orthogonal to our approach.

2.6 Related Work on Online Learning with Multiple Clients

Online learning methods operate on a group of data instances that arrive in a streaming fashion. Most existing work in online learning across multiple clients are within the multi-task learning paradigm. Each client seeks to learn an individual supervised learning model, in conjunction with related clients. The online learning problem with multiple tasks is first introduced by Dekel *et al.* [92]. The relatedness of participated tasks is captured by a global loss and aims to reduce the cumulative loss over multiple rounds. To better model task relationships, Lugosi *et al.* [93] imposes a hard constraint on K simultaneous actions taken by the learner in the expert setting; Agarwal *et al.* [94] uses matrix regularization and Murugesan *et al.* [95] learns task relationship matrix automatically from the data. All these methods are proposed with synchronized protocol and not adaptable for real-world asynchronous learning.

Jin *et al.* [96] presents a distributed framework to perform local training and global learning alternatively with a soft confidence-weighted classifier. Although this is an asynchronous approach, it assumes that the local data is normally distributed, which is not a good fit for non-convex neural network objectives. Besides, it lacks theoretical convergence guarantees and also requires each client to send a portion of its local data to the server.

Different from the above online learning approaches, we design an iterative local computation procedure to balance the previous and current gradients. Besides, we also implement a constrain locally to limit the local deviation and aims to learn an optimal global solution.

2.7 Related Work on Concept Drift

2.7.1 Concept Drift in Non-Distributed Environment

Concept drift is a phenomenon in which the statistical properties of a target domain change over time in an arbitrary way [97]. It was first proposed by Schlimmer et al. [98] who pointed out that noisy data may turn into non-noise information at different time. These changes might be caused by changes in hidden variables which cannot be measured directly

[28].

A wide range of algorithms has been developed to address concept drift. According to the changes in data distribution over time, concept drift can be roughly categorized into two types [99]: 1) sudden/abrupt drift changing data distribution in a quite short period of time; 2) gradual drift which the large data distribution changes has a relatively longer time. There are some other concept drift types such as incremental drift [100], which has a gradual period of "intermediate concept". The term "intermediate concept" was introduced by Gama el al. [31] to describe the transformation between concepts. The intermediate concept of gradual drift is one of the starting or ending concept, while in incremental drift the intermediate concept is a mixture of the starting concept and the ending concept [28]. In this study we focus on the class of sudden concept drift.

Previous work on concept drift usse chunk based learning techniques [101–103], in which a new classifier is learned with a new set of samples. These approaches are not suitable for federated learning, as the sampling rate of device varies, and devices may not generate enough samples quickly enough to build a new classifier. Furthermore, the prediction performance is sensitive to the chunk size, while this parameter is hard to decide in a real-world setting. Other wide used approaches to deal with concept drift problems are ensemble learning-based algorithms, such as using a variant of bagging [104], dynamic adaptation to concept changes (DACC) [105], dynamic weighted majority (DWM) [101] and streaming ensemble algorithm (SEA) [102]. In these algorithms, a new model is created when drift is detected, but this new model is added to an ensemble pool which also includes older models. These ensemble algorithms are not suitable for on-device learning as they cost more to create, train, and deploy.

2.7.2 Dealing with Concept Drift in Distributed Environment

A great many algorithms have explored machine learning algorithms through parallelization and distribution [70,71,106–111]. There are very few approaches that address issues related to concept drift in a fully distributed network. Ang et al. [112] propose a P2P learning

framework for concept drift classification, which includes a drift detection (reactive behavior) and simultaneously a drift prediction (proactive behavior) mechanisms. The basic idea behind the approach is the use of chunk-based technique with a triggering and an ensemble based approaches. As stated in Section 2.7.1, ensemble approaches are not suitable for on-device learning, and moreover, this approach also suffers from high communication cost for the network wide model propagation. Hegedus el al. [113] propose to handle concept drift by maintaining new as well as old models in the network, and models of the data perform random walks in the P2P network. While these techniques can be used in distributed learning, the network topology without a central server is fundamentally different from federated learning.

To the best of our knowledge, currently there are no efficient solution to tackle concept drift efficiently in asynchronous federated learning. We put forward the concept drift issue in the federated learning framework and propose efficient techniques to detect and adapt the sudden drift on local devices.

Table 2.2: Details summary of each real-word time-series dataset.

	ExtraSensory	Air Quality	FitRec
# of tasks	40	9	30
# of time step in each task	3,000	8,218	30,000
Sample rate(s)	20/60	3,600	10
Labels	c	r	r
Dimensionality	276	15	12

2.8 Benchmark Datasets

In this thesis, I perform extensive experiments on the following real-world datasets and benchmark datasets.

- **ExtraSensory Dataset[2]:** A multi-label classification dataset with multiple users, which was previously used for user behavioral context recognition in-the-wild from mobile sensors [114],[55]. Data of each user is collected from mobile device sensors (e.g., high-frequency motion-reactive sensors, location services, audio, watch compass) and watch sensor (accelerator), and follows the non-IID distribution as each user has his/her own devices. We assume that the distribution of data generated by each user's devices is unique to that user and thus is non-IID with respect to the overall distribution across all users. Models of each user are closely related for they have the same target labels (51 activities such as running, walking, etc,.). The label values are binary, 1 indicates certain activity happen and 0 indicates not. We use the provided 225-length feature vectors of time and frequency domain variables generated for each instance. We model device of each user as a client and predict their activities (e.g., walking, talking, running).

- **Air-Quality dataset[3]:** This dataset is from the "2018 KDD CUP of Fresh Air", and data is collected from sensors in weather stations. This is a regression dataset, and the goal is to predict concentration levels of several pollutants (eg,. $PM2.5$, CO, SO_2). The data is recorded every hour, so for each year, there are 8,760 consecutive records. We use data from 9 regions (locations) which are geographical related in Beijing. Data of each region are non-IID because of the geographical distance of the weather sensors. We assume that the distribution of data generated at each weather stations is unique to that region and thus is non-IID with respect to the overall distribution across all regions. We model each region as one client and forecast the values of the

[2]http://extrasensory.ucsd.edu/
[3]https://biendata.com/competition/kdd_2018/data/

target pollutants in the future.

- **FitRec Dataset** [4]**:** User sport records collected from Endomondo, including multiple sources of sequential sensor data such as heart rate, speed, and GPS as well as the sport type (e.g., biking, hiking). Following [115], we re-sampled the data in 10-second intervals, and further generated two derived sequences: derived distance and derived speed. We use data of randomly selected 30 users for heart rate and speed prediction. Similarly to the ExtraSensory Dataset, we model each user as a seperate task (client). We follow the same non-IID assumption as the ExtraSensory dataset as each user has his/her own sensors. The objective is to forecast the users' heart rates and speeds.

- **Fashion-MNIST**[5]**:** This is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes (e.g., Dresses, Coat, Bag). Each class has the same number of examples. We follow a non-IID setting as in *FedAvg* [3] and divide the data into 20 parts according to their labels. We first sort the data by category label, divide each category into 4 different sizes $\{2000, 2750, 3250, 4000\}$, and assign each of 20 parts 2 different sizes. We model each part as a separate client and predict the target labels.

- **Cifar-10**[6]**:** This dataset contains 60,000 images associated with a label from 10 classes (e.g., airplane, dog, cat). Each class has the same number of examples. There are 50,000 training samples and 10,000 test samples. Each image is a 32x32 colour image. We follow a non-IID setting as in *FedAvg* [116] and divide the data into 20 parts according to their labels. We first sort the data by category label, divide each category into 4 different sizes $\{1500, 2250, 2750, 3500\}$, and assign each of 20 parts 2 different sizes. We model each part as a separate client and predict the target labels.

I summarize the detailed statistics of each real-world time-series dataset used in the

[4]https://sites.google.com/eng.ucsd.edu/fitrec-project/home
[5]https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/
[6]https://www.cs.toronto.edu/ kriz/cifar.html

experiments in Table 2.2. c and r show the dataset that is used for classification problems and regression problems in this thesis, respectively.

2.9 Evaluation metrics

The ExtraSensory Dataset (classification) is highly imbalanced and the classification accuracy can be misleading. The rare label appears in 1% of the test set. Hence we report recall, precision, F1 score and balanced accuracy (BA)[55], [114]. For other image data classification, we report the accuracy.

$$\text{Precision} = \frac{1}{K}\sum_{k=1}^{K}\frac{TP^{(k)}}{TP^{(k)}+FP^{(k)}} \tag{2.4}$$

$$\text{Recall} = \frac{1}{K}\sum_{k=1}^{K}\frac{TP^{(k)}}{TP^{(k)}+FN^{(k)}} \tag{2.5}$$

$$\text{Specificity(TNR)} = \frac{1}{K}\sum_{k=1}^{K}\frac{TN^{(k)}}{FP^{(k)}+TN^{(k)}} \tag{2.6}$$

$$\text{F1-score} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{2.7}$$

$$\text{BA} = \frac{\text{Specificity(TNR)}+\text{Recall}}{2} \tag{2.8}$$

where Specificity(TNR) is the true nagative rate, $TP^{(k)}, FP^{(k)}$, $FN^{(k)}, TN^{(k)}$ are the number of truely predicted positive labels, falsely predicted positive labels, falsely predicted negative labels and truely predicted negative labels in user k, respectively.

For the regression dataset we evaluate the performance using Symmetric mean absolute percentage error (SMAPE) and mean absolute error (MAE).

$$\text{MAE} = \frac{1}{KMn_k} \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{i=1}^{n_k} |\hat{y}_i^m - y_i^m| \tag{2.9}$$

$$\text{SMAPE} = \frac{1}{KMn_k} \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{i=1}^{n_k} \frac{|\hat{y}_i^m - y_i^m|}{(\hat{y}_i^m + y_i^m)/2} \tag{2.10}$$

where K is the number of tasks, M, n_k are the number of labels and number of instances, respectively.

Chapter 3: Attention-based Multi-task Learning for Sensor Analytics

In this chapter, I present a novel multi-task deep learning approach, M-Att, which uses attention mechanism to better learn a common feature representation among multiple tasks. Similar to the use of attention networks with encoder-decoder framework for language modeling [16, 17], this approach learns a low-dimensional shared representation for the multiple related tasks. The developed M-Att model uses convolutional neural network (CNN) as encoder and long-short term memory network as decoder for handling multivariate temporal data. To better learn the temporal correlation among related tasks, we employ attention mechanism which seeks to extract useful portion of the representation learned by the encoder CNN model at each task, pay attention to only small amount of information present in multiple inputs and only concentrate on the information related to a specific task at hand. The work presented in this chapter has been published in 2019 IEEE International Conference on Big Data (Big Data).

3.1 Methodology

In this section, we first describe the update process of LSTM and CNN, then we introduce two general multi-task models with no attention mechanisms, finally we explain our proposed M-Att network. In our proposed M-Att, instead of sharing all the features among the different tasks directly, we use the attention mechanism to identify relevant features across multiple tasks. We also combine the CNN and LSTM to capture the local dependencies and long-term dependencies of time series data. In our proposed approach the CNN encoder incorporates the attention mechanism to adaptively select the relevant data series. The

decoder LSTM via temporal attention mechanism selects relevant encoder hidden states across the time steps.

3.1.1 Preliminaries.

Sequence process with LSTM

Long-short term memory (LSTM) [3] captures long-term dependencies issue and models sequential/time series data. Each LSTM unit has a memory cell with the state c_t at time t. Access to the memory cell is controlled by three sigmoid gates: forget gate f_t, input gate i_t and output gate o_t. The update of a typical LSTM cell can be formulated as:

$$i_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{3.1}$$

$$f_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{3.2}$$

$$c_t = i_t * tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) + f_t * c_{t-1} \tag{3.3}$$

$$o_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{V}_o \mathbf{c}_t + \mathbf{b}_o) \tag{3.4}$$

$$h_t = o_t * tanh(c_t) \tag{3.5}$$

where σ represents the sigmoid activation function, $\mathbf{x}_t$ is the input representation at time t, $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_c, \mathbf{W}_o, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_c, \mathbf{U}_o, \mathbf{V}_o$ are learned weight matrices, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o$ are bias vectors. Given a feature representation of task k as $\mathbf{U}_T^{(k)} = \{\mathbf{u}_1^{(k)}, \mathbf{u}_2^{(k)}, ..., \mathbf{u}_t^{(k)}, ..., \mathbf{u}_T^{(k)}\} \in \mathbb{R}^{T \times D}$, the LSTM updates the feature representation with function:

$$\mathbf{h}_t^{(k)} = f(\mathbf{h}_{t-1}^{(k)}, \mathbf{u}_t^{(k)}) \tag{3.6}$$

where $\mathbf{h}_t^{(k)}$ and $\mathbf{h}_{t-1}^{(k)}$ are the hidden states of time step t and $t-1$ for task k, respectively.

$\mathbf{u}_t$ is the input representations at time step t. In the following sections, we use notation $LSTM()$ to represent the whole function update of an LSTM cell for simplify.

Short-term patterns extraction with Convolutional Neural Network[?]

Given the input time series $\mathbf{X}_T^{(k)} = \{\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)}, ..., \mathbf{x}_t^{(k)}, ..., \mathbf{x}_T^{(k)}\} \in \mathbb{R}^{T \times D}$ for each task k, we process them using an encoder CNN, we use a 1D convolutional network at the first layer to extract short-term basic patterns in time dimension and find local dependencies among features. The convolutional layer usually contains n filters, and the output vector of the i^{th} filter at time t is:

$$\mathbf{h}_t^{i(k)} = f(\mathbf{W}_i \mathbf{x}_t^{(k)} + \mathbf{b}_i) \tag{3.7}$$

where $\mathbf{h}_t^{i(k)} \in \mathbb{R}^{D \times L}$, L is the number of filters, $\mathbf{W}_i$ and $\mathbf{b}_i$ are the learnable parameters, and f is the activation function. The temporal attention applied on the concatenated layer after the CNN layer helps to find the feature correlation across all tasks.

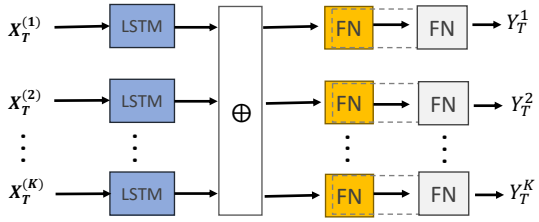


Figure 3.1: Shared LSTM network of Multi-Task Learning (M-LSTM). $\mathbf{X}_T^{(1)}, \mathbf{X}_T^{(2)}, \mathbf{X}_T^{(K)}$ denote the input time series of task 1,2 and K with window size T, respectively. $\mathbf{Y}_T^{(1)}, \mathbf{Y}_T^{(2)}, \mathbf{Y}_T^{(K)}$ are the according labels. FN denotes the fully connected layer. $\oplus$ represents layer concatenation.

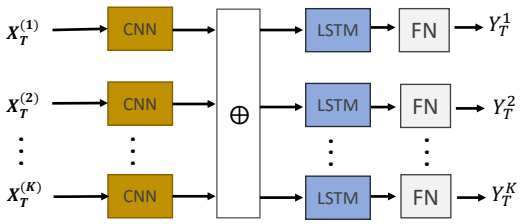


Figure 3.2: Shared CNN/RNN Hybrid network of Multi-Task Learning (M-CNN/LSTM).

3.1.2 Shared LSTM Network.

In this model, for each task k, given input data series $\mathbf{X}_T^{(K)} = (\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)}, ..., \mathbf{x}_T^{(k)})$, we assign a LSTM layer. For general deep multi-task learning framework, there is a shared layer and a private task-specific layer. The shared layer concatenates all the LSTM layer outputs as input. Let $\mathbf{h}_T^{(k)}$ be the output of the k^{th} LSTM layer, the concatenated layer can be represented as $\mathbf{h}_T^{(1)} \oplus \mathbf{h}_T^{(2)} ... \oplus \mathbf{h}_T^{(K)}$, where $\oplus$ represents layer concatenation. As shown in Figure 3.1, tasks share feature information in the shared layer, then go into the task-specific layer separately for prediction.

This model is a naive MTL model because it has only one LSTM layer to learn the feature representation, one shared layer to concatenate the feature patterns and one task-specific layer for output. By taking the output of the shared layer as input for the task-specific layer, this model assumes that tasks are highly related.

3.1.3 Shared CNN/RNN Hybrid Network.

As shown in Figure 3.2, for each input $\mathbf{x}_t^{(k)}$ of task k at time step t, we first use a CNN layer for some basic short-term basic patterns selection [**?**] and get the hidden representation $\mathbf{h}_t^{(k)}$ at time step t.

$$\mathbf{h}_t^{(k)} = f(\mathbf{W}_t \mathbf{x}_t^{(k)} + \mathbf{b}_t) \tag{3.8}$$

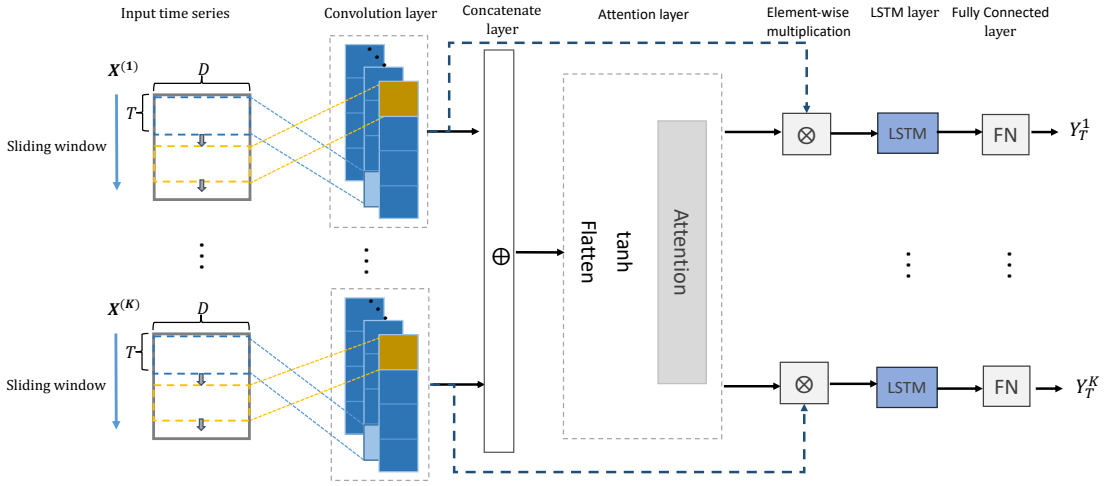


Figure 3.3: Architecture of proposed Attention-based Multi-task learning (M-Att). T is sliding window size, D is feature dimensions. $\oplus$ represents layer concatenation, $\otimes$ indicates element-wise multiplication.

where $\mathbf{W}_t$,$\mathbf{b}_t$ are parameters to learn. Then we concatenate all the hidden representations from the CNNs to get a further concatenated representation $\mathbf{s}_t$.

$$\mathbf{s}_t = \mathbf{h}_t^{(1)} \oplus \mathbf{h}_t^{(2)} ... \oplus \mathbf{h}_t^{(K)} \tag{3.9}$$

The output the shared layer is fed into a higher LSTM layer to learn further hidden representations $\tilde{\mathbf{h}}_t^{(k)}$ of each task k.

$$\tilde{\mathbf{h}}_t^{(k)} = LSTM(\tilde{\mathbf{h}}_{t-1}^{(k)}, \mathbf{s}_t)) \tag{3.10}$$

where $\tilde{\mathbf{h}}_{t-1}$ is the hidden state at time step $t-1$. Finally we predict the target label series with a fully connected layer.

By adding one CNN layer, this model can learn some basic short-term patterns first. This provides useful feature representation for the higher layers.

3.1.4 Attention-based Multi-task Deep Network.

Attention mechanism in deep neural network model can iteratively processes it's input by selecting the relevant content at every time step. In the proposed M-Att approach, the attention mechanism across time dimension is embedded to focus attention on more relevant weights at each time step. Figure 3.3 illustrates the overall network structure of our attention-based multi-task structure. Consider a given task k with D input series, $\mathbf{X}_T^{(K)} = (\mathbf{x}_1^{(k)}, ..., \mathbf{x}_T^{(k)}) \in \mathbb{R}^{D \times T}$, and $\mathbf{x}_t^{(k)} = \{\mathbf{x}_{t1}^{(k)}, \mathbf{x}_{t2}^{(k)}, ..., \mathbf{x}_{tD}^{(k)}\} \in \mathbb{R}^D$ is the input at time step t with dimension D. First we transform each input $\mathbf{x}_t^{(k)}$ using a CNN layer to obtain the hidden representation $\mathbf{h}_t^{(k)}$.

$$\mathbf{h}_t^{(k)} = f(\mathbf{W}_t \mathbf{x}_t^{(k)} + \mathbf{b}_t) \tag{3.11}$$

Then at the last time step T, we get the hidden state $\mathbf{h}_T^{(k)}$ of the k^{th} tasks. Next we concatenate the hidden representation across K tasks to get the shared hidden representation $\mathbf{s}_T$.

$$\mathbf{s}_T = \mathbf{h}_T^{(1)} \oplus \mathbf{h}_T^{(2)} \oplus ... \oplus \mathbf{h}_T^{(K)} \tag{3.12}$$

We pass $\mathbf{s}_T$ to a flatten layer to get a flattened representation $\mathbf{f}$. Then we apply a fully connected layer with T units and $tanh$ nonlinearlity on $\mathbf{f}$ to extract the larger weights and obtain the hidden state $\check{\mathbf{h}}_T \in \mathbb{R}^T$.

$$\mathbf{f} = flatten(\mathbf{s}_T), \check{\mathbf{h}}_T = tanh(\mathbf{f}) \tag{3.13}$$

Then we compute the attention score using a softmax function for each time step $t \in [1, T]$.

$$a_t = \frac{exp(\check{h}_{t\in[1,T]})}{\sum\limits_{t\in[1,T]} exp(\check{h}_t)} \tag{3.14}$$

The attention score a_t is obtained by normalizing the context score $\check{\mathbf{h}}_t$ at each time step t. We then repeat the attention score D times to get the attention vector across input dimension. With permutation we obtain the attention vector $\mathbf{a}_t \in \mathbb{R}^D$ of each task k.

To measure the importance of feature representation at each time step t, we compute the attention vector with an element-wise multiplication of $\mathbf{h}_t^{(k)}$ and the attention vector $\mathbf{a}_t$.

$$\mathbf{c}_t^{(k)} = \mathbf{a}_t \otimes \mathbf{h}_t^{(k)} \tag{3.15}$$

Then we get the extracted hidden representation of each task k at time step t. The obtained hidden vector $\mathbf{c}_t^{(k)}$ is used to update the decoder LSTM:

$$\mathbf{l}_t^{(k)} = LSTM(\mathbf{l}_{t-1}^{(k)}, \mathbf{c}_t^{(k)}) \tag{3.16}$$

With $\mathbf{l}_t^{(k)}$ we can obtain the hidden state $\mathbf{l}_T^{(k)}$ at the last time step T, a following fully connected layer is applied to get the target $\hat{\mathbf{Y}}_T^{(k)}$ of task k.

$$\hat{\mathbf{Y}}_T^{(k)} = \mathbf{W}_T^{(k)}\mathbf{l}_T^{(k)} + \mathbf{b}_T^{(k)}) \tag{3.17}$$

where $\hat{\mathbf{Y}}_T^{(k)}$ is the predicted label matrix, $\mathbf{W}_T^{(k)}, \mathbf{b}_T^{(k)}$ are learnable parameters.

Table 3.1: Parameters settings

Hidden Size (LSTM)	64
Number of filters (CNN)	64
Filter length (CNN)	8
Learning rate (Adam)	0.001
Learning rate (RMSprop)	0.001
Regular dropout	0.2
Recurrent dropout	0.2
Batch size (ExtraSensory)	50
Batch size (Air Quality)	100
Early stop	20

3.2 Experiments

3.2.1 Training Procedure and Hyper-parameters.

For the ExtraSensory Dataset (randomly select 12 users as our dataset), we split each task's first 60% of data for training, the next 20% for validation and the final 20% for testing. And for the Air-Quality dataset, the weather data records from Jan 30 2017 to Jan 31 2018 is split into ratio 3:1 for training and validation, the test data is the weather data from May 1 2018 to May 31 2018.

We use Keras[1] and Tensorflow[2] to implement all our models. The models are trained with backpropagation and Adam optimization algorithm [117] for classification tasks, and RMSprop optimizer [118] for regression tasks. We set the size of the LSTM hidden state as 64, the number of filters for CNN layer as 64 and filter length as 8. We select the batch size of ExtraSensory dataset as 50, and Air Quality dataset as 120.

We performed grid search on the validation set to find the best hyper-parameters. The learning rate for Adam optimizer is selected from $\{0.0001, 0.001, 0.005, 0.01\}$, as for RMSprop, the learning rate is selected from $\{0.001, 0.005, 0.01\}$. To prevent overfitting, we select both the regular dropout and the recurrent dropout from $\{0.1, 0.2, 0.25, 0.3\}$ for LSTM

[1] https://keras.io/

[2] https://www.tensorflow.org/

Table 3.2: Performance comparison of proposed attention based multi-task model (M-Att) and other baseline models on ExtraSensory Dataset. (U_i is user index. *Avg* is the average value across all users. Re, Pr, F1, BA indicate Recall, Precision, F1-score and Balanced accuracy, respectively.)

		U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}	U_{11}	U_{12}	Avg
Re	S-LSTM	0.76	0.56	0.89	0.49	0.62	0.20	0.63	0.64	0.97	0.27	0.78	0.55	0.61
	M-MLP(16,16) [55]	0.60	0.66	0.68	0.25	0.36	0.27	0.31	0.58	0.88	0.40	0.77	0.45	0.52
	M-LSTM	0.64	0.62	0.87	0.59	0.68	0.31	0.33	0.68	0.95	0.59	0.87	0.66	0.65
	M-CNN/LSTM	0.79	0.79	0.88	0.57	0.75	0.56	0.74	0.79	0.84	0.58	0.85	0.60	0.73
	M-Att	0.80	0.78	0.85	0.67	0.55	0.59	0.81	0.70	0.96	0.64	0.87	0.61	**0.74**
Pr	S-LSTM	0.79	0.80	0.90	0.60	0.82	0.38	0.83	0.66	0.98	0.42	0.78	0.67	0.72
	M-MLP(16,16) [55]	0.61	0.69	0.78	0.27	0.41	0.32	0.33	0.50	0.92	0.42	0.78	0.49	0.55
	M-LSTM	0.47	0.48	0.61	0.27	0.32	0.28	0.26	0.44	0.66	0.45	0.58	0.41	0.43
	M-CNN/LSTM	0.44	0.48	0.57	0.30	0.31	0.31	0.35	0.41	0.67	0.38	0.55	0.45	0.45
	M-Att	0.65	0.69	0.84	0.39	0.81	0.61	0.89	0.58	0.96	0.63	0.78	0.68	**0.72**
F1	S-LSTM	0.77	0.66	0.89	0.54	0.71	0.26	0.63	0.65	0.97	0.33	0.78	0.61	0.66
	M-MLP(16,16) [55]	0.60	0.67	0.73	0.26	0.38	0.32	0.33	0.54	0.90	0.41	0.77	0.53	0.53
	M-LSTM	0.54	0.54	0.72	0.36	0.44	0.29	0.29	0.53	0.79	0.51	0.70	0.50	0.52
	M-CNN/LSTM	0.57	0.60	0.69	0.40	0.44	0.40	0.47	0.54	0.75	0.46	0.67	0.52	0.54
	M-Att	0.72	0.74	0.85	0.47	0.65	0.60	0.85	0.63	0.97	0.63	0.82	0.65	**0.72**
BA	S-LSTM	0.87	0.77	0.92	0.73	0.80	0.58	0.75	0.80	0.98	0.61	0.87	0.76	0.79
	M-MLP(16,16) [55]	0.78	0.81	0.83	0.60	0.65	0.61	0.62	0.76	0.94	0.67	0.87	0.70	0.74
	M-LSTM	0.79	0.77	0.91	0.73	0.77	0.61	0.61	0.80	0.97	0.76	0.89	0.78	0.78
	M-CNN/LSTM	0.85	0.84	0.90	0.73	0.79	0.71	0.79	0.85	0.90	0.74	0.87	0.76	0.81
	M-Att	0.88	0.87	0.92	0.79	0.77	0.78	0.90	0.83	0.99	0.80	0.92	0.79	**0.86**

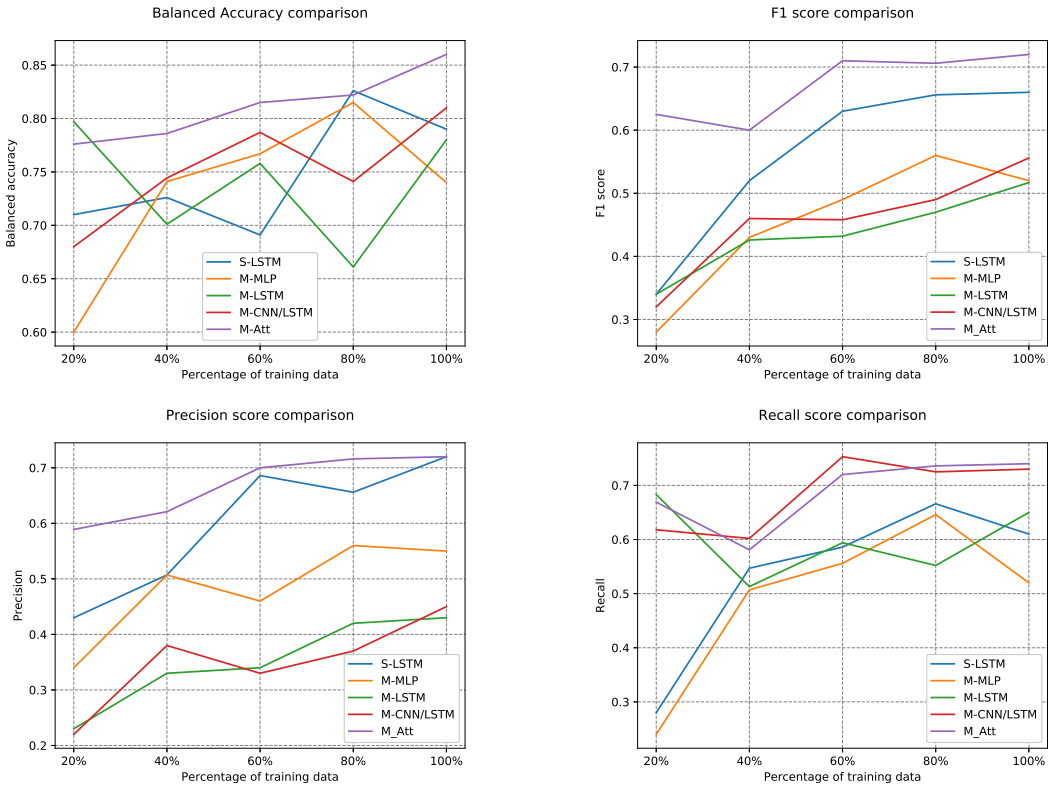


Figure 3.4: Comparison of Balanced Accuracy, F1 socre, Precision and Recall score on all models' change as data size increase.

layer. The best parameters are listed in Table 3.1. We employ early stopping with patience value of 20. For classification tasks, we use cross entropy to monitor the loss. For regression tasks, we change the loss to Mean absolute error (MAE).

3.2.2 Comparative Methods.

We evaluate the performance of our M-Att model with a wide range of baseline models.

- **S-LSTM**. LSTM is a natural selection of sequential data forecasting for its capability of capturing long-term dependencies. In this model, we use a single LSTM layer for single-task prediction.

- **Multilayer Perceptron (MLP(16,16))** [55]. A MLP model with two hidden layers,

where the hidden units used for both layers is 16. This is a multi-task model which learns multiple tasks jointly.

- **M-LSTM**. The Shared LSTM Nextwork in this chapter, we replace the first hidden layer of MLP (16,16) with one LSTM layer which can better capture long-term dependencies in learning. We refer to this model as Multi-task LSTM model (M-LSTM).
- **M-CNN/LSTM**. The Shared CNN/RNN Hybrid Network in this chapter. We refer to this model as multi-task CNN/LSTM model (M-CNN/LSTM).
- **M-Att**. The proposed Attention-based Multi-task Deep Network. We refer to this model as multi-task attention model (M-Att).

3.3 Results and Discussion.

3.3.1 Results of ExtraSensory Dataset.

Table 3.2 shows the multi-label classification performance comparing M-Att model and the other baseline models in terms of Recall, Precision, F1 score and Balanced Accuracy. The M-Att model outperforms all of the other methods on the average performance. Attention mechanism works well on extracting the common features among tasks and achieves the best overall performance on all tasks. Given the highly unbalanced characteristic of the ExtraSensory Dataset, the M-Att model outperforms all the other models in range of 9.1%-38.4% on average F1 score and 6.1%-16.2% on average Balanced Accuracy. The M-CNN/LSTM hybrid model achieves the second best score on average Balanced Accuracy and average Recall score, showing that the CNN layer extracts distinctive features of the correlated multiple time series. However, the feature correlations captured by CNN on each task cannot represent temporal dependency effectively. Also, the shared feature representations across multiple tasks and labels can have negative transfer while using information from rare labels. We observe that M-MLP and M-LSTM have close performance on all evaluation metrics, but not better than the other models. The shared multi-task model

Table 3.3: Comparison of MAE and SMAPE values on proposed M-Att model and other baseline approaches on Air-Quality Dataset. (R_i indicates the i^{th} region of Beijing. Avg is the average value across all tasks.)

		R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	Avg(↓)
MAE	S-LSTM	14.43	10.68	6.67	7.23	7.63	6.36	7.87	12.36	6.10	8.81
	M-MLP(16,16) [55]	8.93	19.84	10.82	9.47	9.19	7.59	**6.56**	11.26	16.35	11.34
	M-LSTM	16.22	13.32	9.21	9.81	9.63	7.99	8.26	9.90	9.82	10.46
	M-CNN/LSTM	13.41	11.73	8.34	8.45	7.80	6.08	6.76	**9.04**	11.56	9.24
	M-Att	**8.82**	**7.34**	**6.61**	**6.90**	**5.13**	**4.35**	6.72	9.49	**4.97**	**6.70**
SMAPE	S-LSTM	0.68	0.51	0.39	**0.49**	0.55	0.41	0.58	0.49	0.39	0.51
	M-MLP(16,16) [55]	0.77	0.71	0.66	0.74	0.74	0.54	0.65	0.60	0.63	0.67
	M-LSTM	0.62	0.71	0.48	0.75	0.50	0.46	0.68	0.59	0.51	0.59
	M-CNN/LSTM	**0.61**	0.63	0.48	0.77	0.59	0.49	0.65	0.63	0.49	0.59
	M-Att	0.73	**0.50**	**0.36**	0.51	**0.42**	**0.36**	**0.54**	**0.42**	**0.36**	**0.46**

without any feature extraction methods will introduce many noise features in the shared representations, thus reducing the overall performance on all tasks.

The single-task learning model (S-LSTM) achieves good performance on Precision and F1 score, but has large fluctuation across different users. However, results of M-Att are stable for most users. The generated shared feature representation by attention mechanism helps each task to get more useful information on prediction, and this is extremely helpful when certain user data lacks training data with regards to a particular class label (activity). For 'U_6' and 'U_{10}', the performance of single-task model is poor, but the the attention-based M-Att model shows high scores on all evaluation metrics. By examining the balanced accuracy of each user across all models in Table 3.2, we notice that there is less fluctuation on each user in the M-Att model. The attention mechanism helps the deep neural network to learn a better feature representation of all tasks.

Learning Curve

Figure 3.4 shows the performance of different models with increasing training data. We used 20%, 40%, 60%, 80% and 100% of each user's training data to test the models' performance change. As training data increases, we see a stable increase with respect to BA, F1 score, Precision and Recall score for the M-Att model. Unstable performance is observed for

the other models. These figures show that attention network can learn a flexible and meaningful feature representation according for different portions of the data; and the model performance is not affected by the data size. Especially, with very small data size, M-Att outperforms the baselines by a significant amount on F1 score and Precision score.

Learning Ability Evaluation

We further evaluate the learning ability of each model by displaying the number of TP (True Positive), FP (False Positve) and FN (False Negative) numbers. As shown in Figure 3.5, M-Att achieves the highest true positive number and lowest false negative number. While S-LSTM has the lower false positive number than M-Att, but also has a lower true positive number than M-Att and M-CNN/LSTM models.

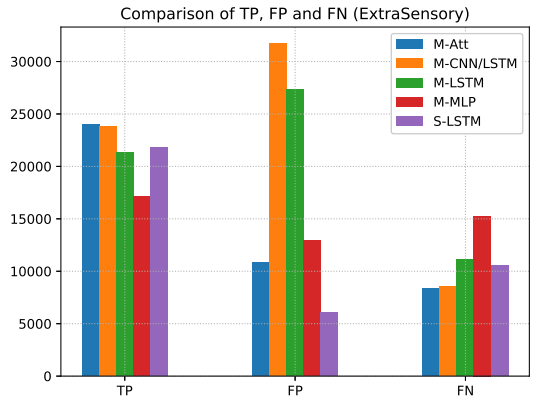


Figure 3.5: Number comparison of TP(True Positive), FP (False Positive), FN (False Negative) for Extrasensory dataset.

3.3.2 Results of Air-Quality dataset.

Table 3.3 reports the performance using the Symmetric mean absolute percentage error (SMAPE) and Mean absolute error (MAE) for all models on Air-Quality dataset. A smaller

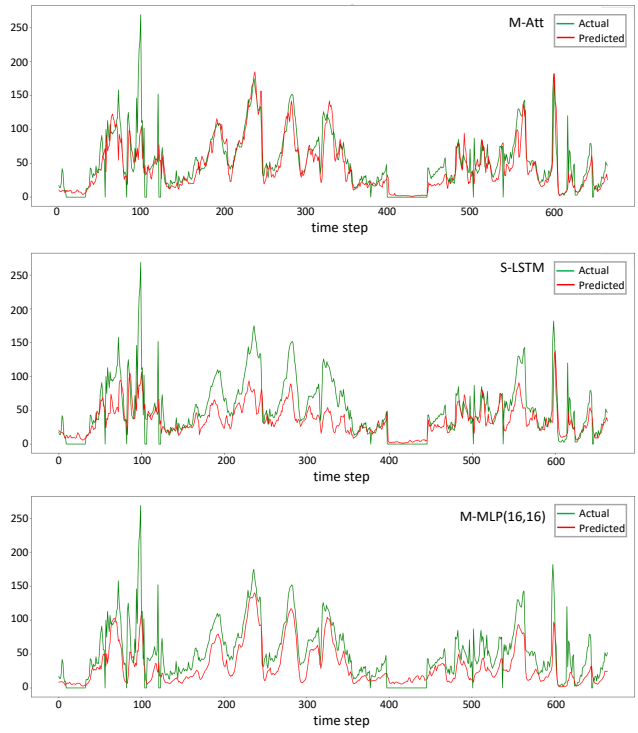


Figure 3.6: Predicted PM2.5 values and actual PM2.5 values with three models (M-Att, S-LSTM and M-MLP(16,16)) for one region of the Air Quality Dataset.

SMAPE or MAE value indicates less difference between the actual value and the predicted value. From the results, we notice that the M-Att model achieves the best performance with the lowest average SMAPE and MAE values, and lower SMAPE values and MAE values for most tasks. This also indicates that with large enough data, the multi-task network with attention mechanism shows promising results for regression data.

Comparison Among Multi-task Models

Comparing the results of the four multi-task models, attention-based multi-task model outperforms the other three. The labels of ExtraSensory Dataset are highly unbalanced and sparse, hence adding the shared layer, each individual task actually benefits with more

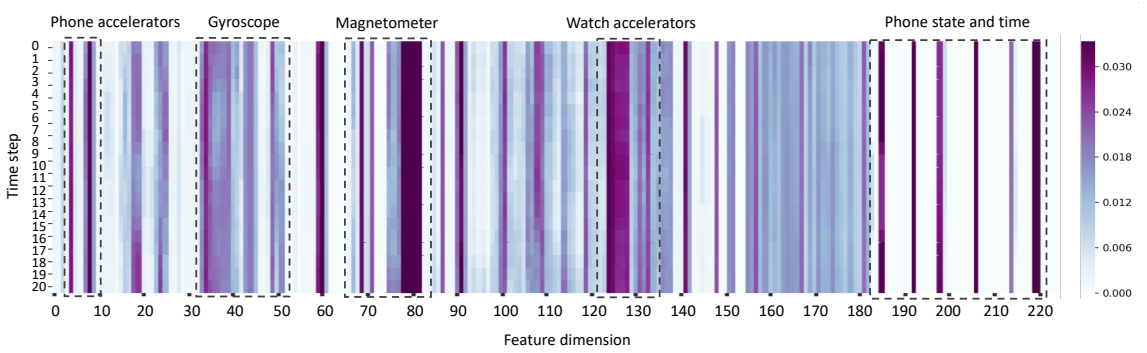


Figure 3.7: Attention weight matrix obtained from attention mechanism with ExtraSensory data, where each row is the attention vector over the input features.

information on labels from other tasks. However, for the Air-Quality dataset, the outputs are consecutive values, the shared layer of LSTM network (M-LSTM), M-MLP and shared CNN/RNN hybrid networks (M-CNN/LSTM) actually result in incorporation of noise from unrelated tasks. M-Att model with attention mechanism can better capture the temporal correlations across tasks.

Prediction Results

Figure 3.6 shows results of the predicted PM2.5 values and actual PM2.5 values of one region from Beijing in the window length of one month. We plot the best performance (M-Att), the second best performance (S-LSTM) and the worst performance (M-MLP(16,16)) for comparison. We observe that M-Att with attention mechanism captures the trend of PM2.5 value change. For most parts, the predicted sequences closely match the trend of the ground-truth.

3.3.3 Evaluation on Attention Mechanism.

To further investigate what attention mechanism captures, we perform a case study on each dataset. First we take one user as an example to visualize the attention vectors for time

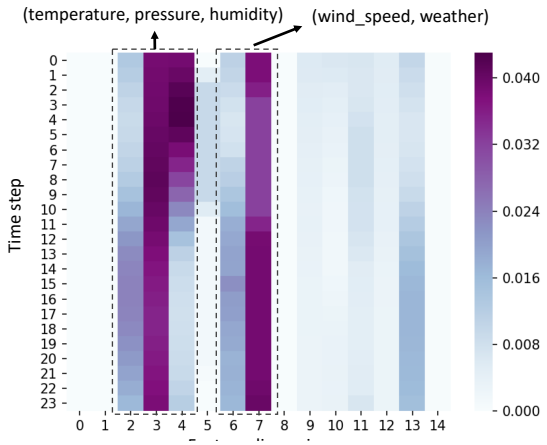


Figure 3.8: Attention weight matrix obtained from attention mechanism with Air Quality data, where each row is the attention vector over the input features.

step window 20 from the ExtraSensory dataset. Figure 3.7 presents the obtained attention matrix, and dark color indicates larger weights. We notice that features of certain sensors (Phone accelerators, Gyroscope, etc,.) have larger weights compared with other inputs, which means that values of these sensors have much tighter correlations with the target activities.

Next we use one region from Air Quality dataset as an example to visualize the attention vectors for a 24 hours time window in Figure 3.8. We observe that the attention weights are evolving across the time steps, because the target air pollutants keep changing in one day length. According to the weight matrix in Figure 3.8, the pressure, humidity and weather condition (sunny, rain, etc,.) are important factors affecting the concentration of air pollutants, especially the pressure and weather condition. We also find that the effect of humidity is decreasing as the time step window increases since humidity is affected by weather condition, so it is not a direct indicator to the air pollutants. Temperature and weather are closely related to the readings of air pollutants, because humans in northern area burn fuel (coal) for heating in winter time. All these facts demonstrate that the attention successfully captures the feature representations among tasks, and the correlations between

the input features and target series.

3.4 Summary

In this work I proposed a novel attention network M-Att for multi-task learning with sensor data. I also developed two other shared deep neural networks (M-LSTM and M-CNN/LSTM) for comparison. I applied our proposed multi-task models on classification and regression tasks. The experiments show that the proposed M-Att with attention mechanism extracts shared feature representations across tasks with high flexibility. Especially in the situation when a certain task lacks training data, the attention-based multi-task model can achieve much better performance. The proposed M-Att model is robust for the multi-label and highly unbalanced class label benchmarks. In addition, I visualize the attention weights to show the interpretation of our approach.

Chapter 4: Federated Multi-task Learning with Hierarchical Attention for Sensor Data Analytics

In this chapter, I present a federated multi-task learning framework. This proposed approach uses a hierarchical attention mechanism to learn a common/shared representation among multiple tasks. The inter-feature correlations of each individual task are captured by sensor-specific attention layers applied on input sensors' time series. Meanwhile, the temporal correlations are captured by attending to all tasks across the time dimension. The scope of this method involves modeling the input from multiple tasks, where each task has multiple sensors. The proposed algorithm is federated in the sense that no data leaves the task. However, I assume that data from each of the tasks are available at the same time. This model has been evaluated on both classification and regression tasks. The prediction results show that our proposed federated multi-task hierarchical attention model (FATHOM) outperforms state-of-the-art baselines. The work presented in this chapter has been published in International Joint Conference on Neural Networks (IJCNN 2020).

4.1 Methods

For multi-label classification problems, we adopt the loss function from [119]; that seeks to prevent over-fitting and makes the model more adaptable with an additional regularization component added to the cross-entropy loss. This is given as:

$$\mathcal{L}(\hat{\mathbf{Y}}, \mathbf{Y}) = -(1-\alpha) \underbrace{\sum_{m=1}^{M} \sum_{t=1}^{n_k} y_t^m \log(\hat{y}_t^m)}_{a} - \frac{\alpha}{M} \underbrace{\sum_{m=1}^{M} \sum_{t=1}^{n_k} \hat{y}_t^m}_{b}, \tag{4.1}$$

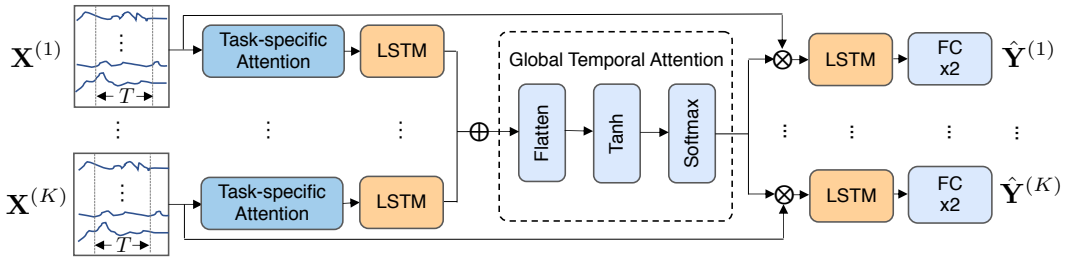


Figure 4.1: Architecture illustration of the proposed federated Multi-task Hierarchical Attention Model (FATHOM). $\mathbf{X}^{(1)}, ..., \mathbf{X}^{(K)}$ indicate input data series of K tasks, T is the sliding window size, and $\hat{\mathbf{Y}}^{(1)}, ..., \hat{\mathbf{Y}}^{(K)}$ are the predicted labels. 'FCx2' indicates two fully connected layers, $\otimes$ indicates element-wise multiplication, $\oplus$ indicates tensors' concatenation.

where $\hat{y}_t^m, y_t^m$ are the predicted and true label for the m^{th} label at time step t, respectively. The part a of Equation 4.1 is the cross-entropy loss, and the second part b is the added uniform distribution, which is a measure of how dissimilar the predicted distribution is to uniform. $\alpha \in [0, 1]$ is an adjustable parameter and can be changed to control the amount of uniform distribution added. We performed a grid search on a validation set and found a value of $\alpha = 0.3$ showed the best performance across the benchmarks studied in this chapter.

4.1.1 Model Structure

The proposed model includes two main components: 1) task-specific local attention to learn feature representations of each device and 2) global temporal attention to extract a shared temporal representation across all devices. An illustration of our model structure can be found in Figure 4.1. $\mathbf{X}^{(1)}, ..., \mathbf{X}^{(K)}$ are the input data series of K devices. For each device k, we use a sliding window of size T to process the sequential data. A task-specific attention layer is applied on each input layer to capture local feature dependencies of each device. After task-specific attention, the obtained attention vector is fed into the first LSTM layer to further learn a local feature representation. The computational process of this part is shown in lines 1-4 of Algorithm 1. Then we pass the concatenated local feature representations

Algorithm 1 Learning for FATHOM

Input: $\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, ..., \mathbf{X}^{(K)}$ K tasks, stored on K nodes separately

1: **for** $k \in \{1, 2, ..., K\}$ in parallel over K nodes **do**
2: calculate attention vector $\boldsymbol{\phi}_d^{(k)}$ and iterate each feature to get a matrix $\boldsymbol{\Phi}^{(k)}$
3: pass $\boldsymbol{\Phi}^{(k)}$ to a LSTM layer to get hidden representation $\mathbf{h}_T^k$
4: Central node calculates $\mathbf{s}_T \leftarrow \mathbf{h}_T^1 \oplus \mathbf{h}_T^2 ... \oplus \mathbf{h}_T^K$
5: Compute global attention $\mathbf{a}_{1 \rightarrow T}$ based on $\mathbf{s}_T$
6: **for** $k \in \{1, 2, ..., K\}$ in parallel over K nodes **do**
7: update attention vector $\boldsymbol{\psi}_t^{(k)} \leftarrow \mathbf{x}_t^{(k)} * \mathbf{a}_t^{(k)}$
8: **return** $\hat{\mathbf{Y}}^{(1)}, \hat{\mathbf{Y}}^{(2)}, ..., \hat{\mathbf{Y}}^{(K)}$ K label matrices

to the global temporal attention to learn a shared temporal representation (found in Lines 5-6 of Algorithm 1). The shared temporal representation is sent back to each device. For each device, by performing the element-wise multiplication of raw input features and the shared attention weights, we obtain the updated temporal attention vector for each device (lines 7-9 of Algorithm 1). A second LSTM layer is applied to each device after the global temporal attention layer. Finally, a classifier is used to predict the labels for each device. In the following sections we describe each part in detail.

Feature Representation learning with LSTM

We use a Long Short-Term Memory (LSTM) layer to process the generated feature representations from the task-specific attention and global temporal attention. Consider a feature representation of task k as $\mathbf{E}^{(k)} = \{\mathbf{e}_1^{(k)}, \mathbf{e}_2^{(k)}, ..., \mathbf{e}_t^{(k)}, ..., \mathbf{e}_T^{(k)}\} \in \mathbb{R}^{T \times D}$, the LSTM updates the feature representation with function:

$$\mathbf{h}_t^{(k)} = f(\mathbf{h}_{t-1}^{(k)}, \mathbf{e}_t^{(k)}) \tag{4.2}$$

where $\mathbf{h}_t^{(k)}$ and $\mathbf{h}_{t-1}^{(k)}$ are the hidden states of time step t and $t-1$ for task k, respectively. $\mathbf{e}_t$ is the input at time step t. We adopt the LSTM structure from [12]:

$$i_t = \text{sigmoid}(\mathbf{W}_i \mathbf{e}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{4.3}$$

$$f_t = \text{sigmoid}(\mathbf{W}_f \mathbf{e}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{4.4}$$

$$c_t = i_t * \tanh(\mathbf{W}_c \mathbf{e}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) + f_t * c_{t-1} \tag{4.5}$$

$$o_t = \text{sigmoid}(\mathbf{W}_o \mathbf{e}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{V}_o \mathbf{c}_t + \mathbf{b}_o) \tag{4.6}$$

$$h_t = o_t * \tanh(c_t) \tag{4.7}$$

where c_t is the cell state at time t. i_t, f_t, o_t are sigmoid gates of input, forget, and output, respectively. $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_c, \mathbf{W}_o, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_c, \mathbf{U}_o, \mathbf{V}_o$ are learned weight matrices, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o$ are bias vectors. For simplicity, we use notation LSTM() to represent the whole function update of an LSTM cell in the following sections.

Task-specific Attention.

Feature extraction at the input task level increases the probability of capturing task-related features. These features should be given higher weights in comparison to other features while computing a task-specific representation. However, it is unknown which part of the feature space has predictive information, so we choose a soft attention mechanism [57] to capture feature representations by attending to all input features from each task.

Consider $\underline{\mathbf{X}}^{(k)} = \{\mathbf{x}_{*,1}^{(k)}, ..., \mathbf{x}_{*,d}^{(k)}, ..., \mathbf{x}_{*,D}^{(k)}\} \in \mathbb{R}^{T \times D}$ as an input example for a given task k with D input series, where T is the time window size, and $\mathbf{x}_{*,d}^{(k)} \in \mathbb{R}^T$ is the d-th column in $\underline{\mathbf{X}}^{(k)}$. First we transform $\underline{\mathbf{X}}^{(k)}$ using a fully connected layer to obtain a hidden representation $\mathbf{Z}^{(k)}$ as:

$$\mathbf{Z}^{(k)} = \underline{\mathbf{X}}^{(k)} \mathbf{W}^{(k)} \in \mathbb{R}^{T \times D}, \tag{4.8}$$

where $\mathbf{W}^{(k)}$ is the trainable weight matrix. Let $\mathbf{z}_d^{(k)}$ be the d-th column of $\mathbf{Z}^{(k)}$, we compute the attention weight of the d-th input series by applying a softmax function:

$$\mathbf{a}_d^{(k)} = \frac{\exp(\mathbf{z}_d^{(k)})}{\sum_{g=1}^{D} \exp(\mathbf{z}_g^{(k)})} \in \mathbb{R}^T. \tag{4.9}$$

We measure the importance of features by computing the context vector with the element-wise multiplication of $\mathbf{x}_{*,d}^{(k)}$ and the attention weights $\mathbf{a}_d^{(k)}$. Then a tanh activation is applied to obtain the final attention vector at dimension d:

$$\boldsymbol{\phi}_d^{(k)} = \tanh(\mathbf{x}_{*,d}^{(k)} \otimes \mathbf{a}_d^{(k)}) \in \mathbb{R}^T. \tag{4.10}$$

Using eq. 4.10, we iterate each feature to get a matrix $\boldsymbol{\Phi}^{(k)} \in \mathbb{R}^{T \times D}$ across feature dimension D. Then we use a LSTM to update each row in $\boldsymbol{\Phi}^{(k)}$ get the hidden representation $\mathbf{h}_t^{(k)}$ at time step t:

$$\mathbf{h}_t^{(k)} = \text{LSTM}(\mathbf{h}_{t-1}^{(k)}, \boldsymbol{\Phi}_t^{(k)}). \tag{4.11}$$

Global Temporal Attention.

The attention distribution captured at task-specific levels focuses on a specific part of features for individual devices, which can only reflect the label information at a current time window. However, for time series data, there is usually a strong temporal correlation. Hence, it is essential to capture the temporal dependencies across time. The global attention component aims at learning a shared representation across all tasks at each time step. For task k, with iteration of $\mathbf{h}_t^{(k)}$ through the time window T we obtain the hidden representation $\mathbf{h}_T^{(k)}$ after the first LSTM layer. First we concatenate the hidden representation

across K devices to get the shared hidden representation $\mathbf{S}_T$:

$$\mathbf{S}_T = \mathbf{h}_T^{(1)} \oplus \mathbf{h}_T^{(2)} \oplus ...\mathbf{h}_T^{(k)}... \oplus \mathbf{h}_T^{(K)}. \tag{4.12}$$

We pass the shared hidden representation to a flatten layer to get the hidden representation $\mathbf{f}$. Different from task-specific attention, we apply a tanh nonlinearlity before softmax. By transforming $\mathbf{f}$ using a fully connected layer with T units, tanh is applied to obtain the time-step level context vector $\mathbf{u}_T$ by $\mathbf{u}_T = \tanh(\mathbf{f}) \in \mathbb{R}^T$.

Then we compute the global temporal attention score using a softmax function for each time step $t = 1, .., T$:

$$a_t = \frac{\exp(u_{t \in T})}{\sum\limits_{j \in T} \exp(u_j)}. \tag{4.13}$$

The attention score a_t is obtained by normalizing the context score u_t at each time step t. We then iterate each feature d for a_t, and obtain the attention vector $\mathbf{a}_t \in \mathbb{R}^D$.

We measure the importance of each time-step by computing the attention vector with an element-wise multiplication of $\mathbf{x}_t^{(k)}$, which is the t-th row of $\underline{\mathbf{X}}^{(k)}$, and the attention vector $\mathbf{a}_t$:

$$\boldsymbol{\psi}_t^{(k)} = \mathbf{x}_t^{(k)} \otimes \mathbf{a}_t \in \mathbb{R}^D. \tag{4.14}$$

Here we obtain the extracted hidden representation for each task k at the time step t. Then we feed $\boldsymbol{\psi}_t^{(k)}$ of task k to a LSTM layer and get the hidden representation at time step t:

$$\mathbf{h'}_t^{(k)} = \text{LSTM}(\mathbf{h'}_{t-1}^{(k)}, \boldsymbol{\psi}_t^{(k)}). \tag{4.15}$$

Finally, we iterate each time step and get the hidden state $\mathbf{h'}_T^{(k)}$ at the last time step T. This hidden state is fed into two fully connected layers to get the predicted labels as below:

$$\tilde{\mathbf{h}}_T^{(k)} = \mathbf{W}_T^{(k)} \mathbf{h}'^{(k)}_T + \tilde{\mathbf{b}}_T^{(k)}, \tag{4.16}$$

$$\underline{\hat{\mathbf{Y}}}^{(k)} = \mathbf{V}_T^{(k)} \tilde{\mathbf{h}}_T^{(k)} + \mathbf{b}_T^{(k)}, \tag{4.17}$$

where $\tilde{\mathbf{h}}_T^{(k)}$ is the hidden state after the first fully connected layer, $\underline{\hat{\mathbf{Y}}}^{(k)}$ is the predicted labels of $\underline{\mathbf{X}}^{(k)}$, $\mathbf{W}_T^{(k)}, \mathbf{V}_T^{(k)} \in \mathbb{R}^{D \times T}, \tilde{\mathbf{b}}_T^{(k)}, \mathbf{b}_T^{(k)} \in \mathbb{R}^T$ are the learned parameters.

4.2 Experiments

4.2.1 Comparative Methods

We compare the proposed FATHOM approach to several single-task learning and multi-task learning approaches. We select the following state-of-the-art approaches as baselines:

- **Logistic Regression (LR)** [56]. This is a single task learning approach. Each task performs training and prediction with its own data. There is no parameter sharing among any tasks.
- **Multilayer Perceptron (MLP(16,16))** [55]. A multi-task MLP model with two hidden layers, where the hidden dimension used for both layers is 16.
- **Convolutional Recurrent Neural Network (CRNN)** [120]. A multi-task learning model that uses Convolutional Neural Networks to extract short-term basic patterns and find local dependencies among features.
- **M-Att** [32]. A hybrid multi-task attention model with a combination of Convolutional Neural Networks and Recurrent Neural Networks.
- **S-LSTM** A single task learning model with a single LSTM layer as a comparison with the LR approach.
- **M-LSTM**. A multi-task model with the first hidden layer of MLP(16,16) replaced by one LSTM layer which can better capture long-term dependencies in learning.

In particular, we perform ablation studies to assess the strengths of the different attention layers introduced in our proposed FATHOM.

- **FATHOM-ta**. FATHOM without the task-specific attention. This model is used to measure the importance of global temporal attention.
- **FATHOM-ga**. FATHOM without the global temporal attention. This model is used to measure the learning ability of task-specific attention.

4.2.2 Evaluation Metrics and Setup

We compare the performance of these models with the proposed FATHOM model. For the classification dataset, the labels are highly imbalanced and hence we report the F1 score, precision, recall, and Balanced Accuracy (BA) [55]. For the regression datasets, we evaluate performance using symmetric mean absolute percentage error (SMAPE) and mean absolute error (MAE).

Training Setup.

For each experiment, we split our dataset into training data, validation data and test data, in the proportions of 60%, 20%, and 20%, respectively. We use Keras and Tensorflow to implement all the approaches. Our models are trained with Adam optimizer for classification tasks, and RMSprop for the regression tasks.

Hyper-parameters.

Based on the performance on the validation set we choose the best group of parameters, retrain a model with the identified parameters, and report results on the test set. We set the hidden units of both LSTM layers to 64, and both the regular dropout and the recurrent dropout to 0.2. We also impose $l2$ constraints with value 0.001 on the weights within LSTM nodes to further reduce over-fitting. We use a batch size of 60 for ExtraSensory classification tasks, and 100 for the other two regression datasets. The initial learning rates are set to 0.001.

4.3 Results

4.3.1 Comparative Performance

We demonstrate the prediction performance of the proposed approach in comparison to different baseline approaches. Table 4.1 shows the performance for the different models on ExtraSensory, Air Quality and FitRec Datasets.

For the classification performance on ExtraSensory Data, we observe that FATHOM significantly outperforms the other models in terms of precision, recall, F1, and Balanced Accuracy metrics. Given the highly imbalanced distribution for ExtraSensory data, FATHOM outperforms all baseline models in the range of 10.81%-57.69% on F1 score and 4.76%-22.22% on Balanced Accuracy. We observe that CRNN performs slightly better than multi-task LSTM (M-LSTM) and LR approaches, but not better than other models. The feature correlations captured by CNN on each task is loose and cannot represent temporal dependency effectively. M-Att model with one attention layer capturing the temporal correlations among input sequences achieves close performance as FATHOM-ta.

From the regression results, we observe that FATHOM outperforms all baseline models in SMAPE by a range 15.38%-64.22%, and in MAE by a range 12.04%-75.77% for the Air Quality dataset. For FitRec, FATHOM achieves the best performance for three out of four results. All results on three datasets show that our model achieves the best performance, which indicates the effectiveness of the proposed method in both classification and regression problems.

Ablation Study.

To further evaluate the two attention mechanisms in FATHOM we perform an ablation study by removing either the task-specific attention layer or the global temporal attention layer and denote them by FATHOM-ta and FATHOM-ga, respectively. From Table 4.1, we observe that FATHOM-ta with the global temporal attention still achieves very good performance in comparison to other baseline approaches. However, FATHOM-ga does not

Table 4.1: Comparative Performance on three datasets. 'HRate' represents 'Heart Rate'. 'Improv.(a)' and 'Improv.(b)' show the percentage improvement of FATHOM over the worst and best baseline results, respectively. * indicates significantly better than the second best score ($p < 0.05$)

Methods	ExtraSensory				Air Quality		FitRec			
	Pr↑	Re↑	F1↑	BA↑	mae↓	smape↓	mae↓ (HRate)	mae↓ (Speed)	smape↓ (HRate)	smape↓ (Speed)
LR	0.57	0.60	0.52	0.72	30.13	1.23	14.56	13.86	0.64	0.61
MLP(16,16)	0.55	0.61	0.58	0.76	10.83	0.65	7.98	8.88	0.37	0.41
CRNN	0.43	0.68	0.54	0.78	10.43	0.64	9.05	9.38	0.35	0.35
M-Att	0.69	0.72	0.70	0.83	8.30	0.46	4.62	4.71	0.30	0.32
S-LSTM	0.79	0.71	0.74	0.84	9.11	0.52	13.51	12.96	0.58	0.58
M-LSTM	0.45	0.62	0.52	0.77	10.39	0.66	7.0	6.93	0.35	0.29
FATHOM-ga	0.50	0.61	0.54	0.77	15.67	0.73	7.59	7.11	0.34	0.31
FATHOM-ta	0.80	0.69	0.74	0.84	8.15	0.51	4.18	4.10	**0.19**	0.20
FATHOM	**0.89***	**0.77***	**0.82***	**0.88***	**7.30***	**0.44***	**3.93***	**4.0***	0.20	**0.17***
Improv.(a)%	106.97	28.33	57.69	22.22	75.77	64.22	70.91	69.13	68.75	70.69
Improv.(b)%	12.65	6.94	10.81	4.76	12.04	15.38	14.93	15.07	33.33	41.37

perform well, because the feature representations learned by task-specific layer fail to leverage the temporal correlations in the input data. The FATHOM model with both attention mechanism outperforms FATHOM-ga and FATHOM-ta by 51.85% and 10.81% in terms of F1-score on Extrasensory , respectively. We also observe that FATHOM-ta is close in performance for the other values with FATHOM. This is because training data of each user in FitRec is biased to one sport type (e.g., biking, hiking), therefore a shared global temporal feature representation is important for future prediction.

Single-task versus Multi-task learning.

We assess the benefits of multi-task learning in comparison to single-task learning models (See Table 4.1). The single-task LR model has the worst performance on all three datasets. The single task S-LSTM model that captures temporal dependencies outperforms the LR model. However, the performance of jointly trained multiple task learning approaches with LSTM (M-LSTM) is worse compared to the S-LSTM model for ExtraSensory and Air Quality datasets. As mentioned before, for the FitRec dataset, the single task models get

the worst performance because the training data for each task is imbalanced, so the model performance will be harmed if the user switched to another activity in test data. This in turn shows the benefit of multi-task learning. In general, multi-task learning approaches improve classification/regression performance but fail when the relationships among multiple tasks are not modeled well. FATHOM, on the other hand, outperforms the single task learning models because it is able to identify specific key features across different tasks and across different time steps.

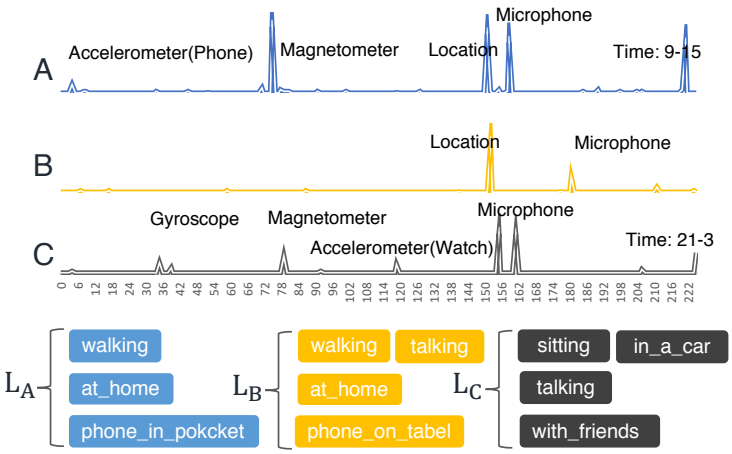
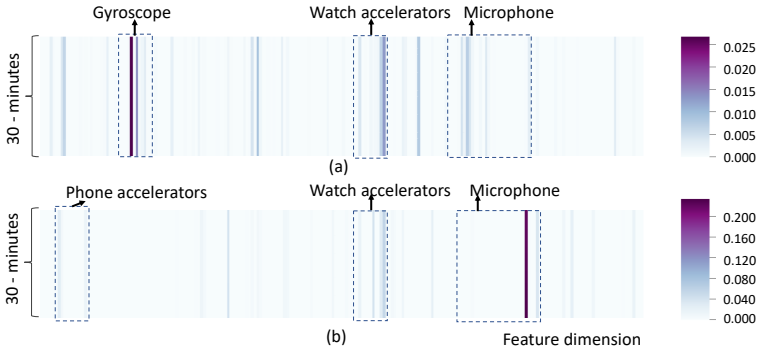


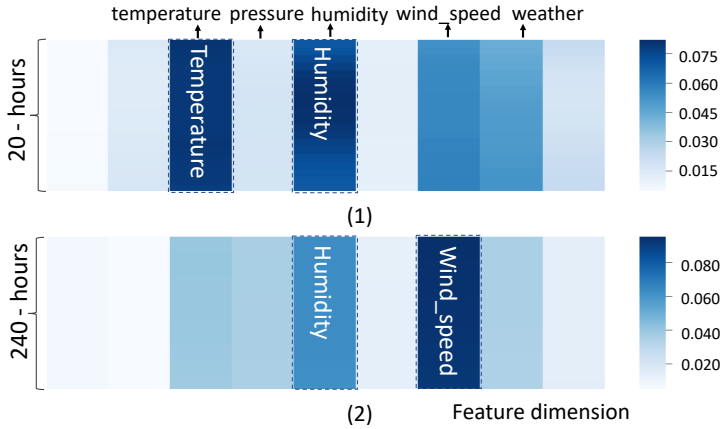
Figure 4.2: Attention weights in feature dimensions captured with task-specific attention in ExtraSensory dataset. (a): A, B, C represent three different users. L_A, L_B, L_C represent their according labels.

4.3.2 Task-specific Attention Evaluation

To better understand the attention mechanisms and their abilities, we present several qualitative studies. Figure 4.2 shows the burstiness of features (spikes) captured by the task-specific attention from three different tasks at different time steps of ExtraSensory dataset. We observe a high correlation between the feature spikes and the corresponding labels. For example, for person A who is walking at home with a phone in pocket; the captured



(a) ExtraSensory



(b) Air Quality

Figure 4.3: Sensor-specific attention matrix from the ExtraSensory Dataset (a) and Air Quality dataset (b). (a): predictions of 30-minutes time length of two users. (b) predictions of one task in 20-hours and 240-hours time length. Each column is the attention vector over the input series.

related features are phone accelerometers, magnetometer, location, microphone, and time. For person B, who is walking but talking to a phone on the table, there is no change of the magnetometer, no acceleration of the phone, and also a lower volume of voice. For person C who is driving a car and talking with friends, the task-specific attention captures the correlated features to corresponding group activities.

We take one task from the Air Quality dataset and two tasks from the ExtraSensory dataset to further visualize the variation of attention vectors across feature dimensions in Figure 4.3. From the attention weight matrix shown in Figure 4.3(a) we observe that the

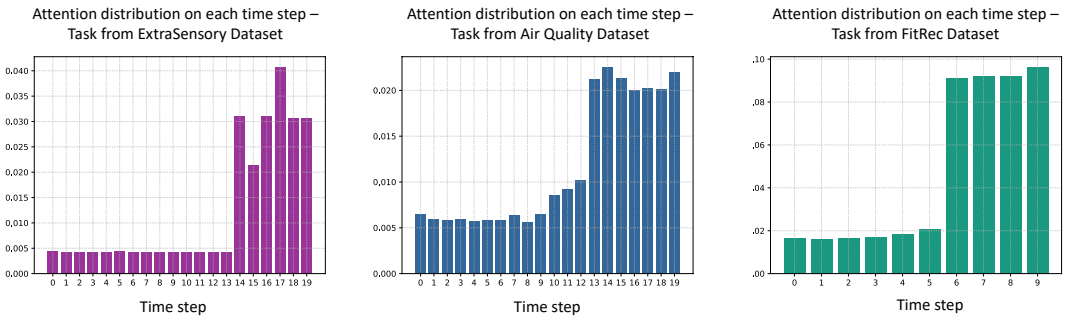


Figure 4.4: Time-dimension attention distribution of three different tasks

user of matrix (1) first lies down, then walks and talks with friends on a phone in pocket. The captured highly related features are phone gyroscope, watch accelerator, and microphone. The user of matrix (2) first grooms and gets dressed, then stays in a lab. We find that watch and phone accelerators have a strong correlation with body movement. The microphone is directly correlated with voice in the surroundings. Figure 4.3(b) represents that in a prediction window of 20 hours length, temperature and humidity have the highest weights among all the input features. Recall that the attention weights semantically indicate the relative importance of each local feature. We find that in the case of short term prediction, temperature and humidity affect the air pollutants most. This is because in the winter users in Beijing consume fuel for heating and humidity is usually low. While in a prediction window length of 240 hours, wind speed becomes the most important with the highest weight. In a dry season with low temperatures, the only effective way to disperse air pollutants is wind. All the above case studies show that our method is effective at capturing task-specific features that vary across time scales leading to interpretable results.

4.3.3 Global Attention Evaluation

Figure 4.4 shows the attention distribution from the central time attention layer of one task from the three datasets, respectively. The LSTM layer will allocate the weights to the last one or two time steps, thus the information of former steps will be lost. By applying the central time attention, the weight distribution does not just focus on the last step, but also

spreads to former steps. My observation is that temporal information is not lost and gets re-introduced leading to a stronger predictive performance.

4.4 Summary

In this chapter I present FATHOM, a novel federated multi-task model utilizing a hierarchical attention mechanism to generate more efficient device-specific feature representations. Task-specific attention is designed to capture feature correlations within each local task and global temporal attention is used to generalize inter-task feature representations across all tasks. The proposed model is evaluated on both classification and regression tasks. The results show that this approach improves prediction performance significantly compared to a wide range of state-of-the-art methods. I also show multiple qualitative case studies to interpret the attention mechanisms in our model. However, the proposed method works in a synchronous fashion.

Chapter 5: Asynchronous Online Federated Learning for Edge Devices

In this chapter, I present an asynchronous online federated learning framework (ASO-Fed), where the central model does not wait for collecting and aggregating the gradient information from lagging clients, and clients perform online learning to deal with local streaming data. By design, ASO-Fed enables wait-free computation and communication, which ensures the model always converges better than synchronized FL frameworks. This study focuses on improving prediction performance and computation efficiency in FL instead of communication costs or privacy issues. ASO-Fed shares similar privacy benefits as other general FL algorithms [4, 5, 20, 116] as the data does not leave the edge devices.

In practice, I find that ASO-Fed is particularly useful for streaming data with heterogeneous devices having different computing/communication speeds. Besides the prediction performance, I also simulate different network delays for each device to show the computational efficiency of ASO-Fed. The work presented in this chapter has been published in IEEE International Conference on Big Data (Big Data 2020).

5.1 Preliminaries

In this section, we first present the general form of federated learning. Then we briefly introduce the commonly used *FedAvg* [3] and identify the issues in synchronized federated settings.

We define local empirical loss of client k as:

$$f_k(w_k) \stackrel{def}{=} \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \ell_i(x_i; w_k), \tag{5.1}$$

where $\ell_i(x_i; w_k)$ is the corresponding loss function for data point $\{x_i, y_i\}$. We denote $N = \sum_{k=1}^{K} n_k$ as the total number of samples across K devices. We can obtain the following global objective function:

$$F(w) = \sum_{k=1}^{K} \frac{n_k}{N} f_k(w) = \sum_{k=1}^{K} \frac{n_k}{N} \cdot \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \ell_i(x_i; w). \tag{5.2}$$

where w is the aggregated global model [1]. The overall goal is to find a model w_* with:

$$w_* = \arg\min F(w) \tag{5.3}$$

5.1.1 Synchronized Federated Optimization

As shown in Algorithm 2, for *FedAvg*, at each global iteration, a subset of the devices are selected and run *SGD* locally to optimize the local objective function f_k on device k, and then communicate their local model updates to the server for aggregation.

With heterogeneous local objectives f_k, carefully tuning of local epochs is crucial for *FedAvg* to converge, however, a larger number of local epochs may lead each device towards the optima of its local objective as opposed to the global objective. Besides, data continues to be generated on local devices, increasing local variations relative to the global model. Therefore, we incorporate a constraint to restrict the amount of local deviation by penalizing large changes from the current model at the server. We explain this in detail in Section 5.2.2.

Most synchronized federated optimization methods have a similar update structure as *FedAvg*. One apparent disadvantage of this structure is that, at each global iteration, when one or more clients are suffering from high network delays or clients which have more data and need longer training time, all the other clients must wait. Since the server aggregates after all clients finish, the extended period of waiting time in a synchronized optimization

[1] We use w and w_k to represent the central model and client model, respectively.

Algorithm 2 Algorithm for FedAvg

Input: K indexed by k, local minibatch size B, local epochs E and learning rate η.
Central Server:
for global iterations $t = 1, 2, ..., T$ **do**
    Server chooses a subset S_t of K devices at random
    for each client $k \in S_t$ in parallel **do**
        $w_k^{t+1} \leftarrow \text{ClientUpdate}(k, w^t)$
    $w^{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{N} w_k^{t+1}$
ClientUpdate(k, w^t)**:**
device k updates w^t for E epochs of SGD on f_k with η
return w_k^{t+1} to server

protocol will lead to idling and wastage of computing resources [121, 122].

5.2 Proposed Method

We propose to perform asynchronous online federated learning where the server begins to update the central model w after receiving updates from one client, without waiting for the other clients to finish. The server maintains the current central model w, while all clients maintain their own copies (w_k) of w in the memory. Note that the copy of w at one client may be different from the copies at other clients.

Figure 5.1 illustrates the update procedure for ASO-Fed. The server starts aggregation after receiving one client's update, and performs feature learning on the aggregated parameters to extract a cross-client feature representation. Then the server starts the next iteration and distributes the new central model to the ready clients. Clients may have new data samples during the training process. To better capture the inter-client relatedness, we use a decay coefficient to balance the previous and current local gradients with an iterative local computation procedure. The approach of ASO-Fed is detailed in Algorithm 3. We will explain each part in detail in the following sections.

As an example in Figure 5.1, when the server node receives the gradient uploaded from the lagging clients (e.g., Client 2), it has already updated the central model twice. We can observe that there is an inconsistency in the asynchronous update scheme when it

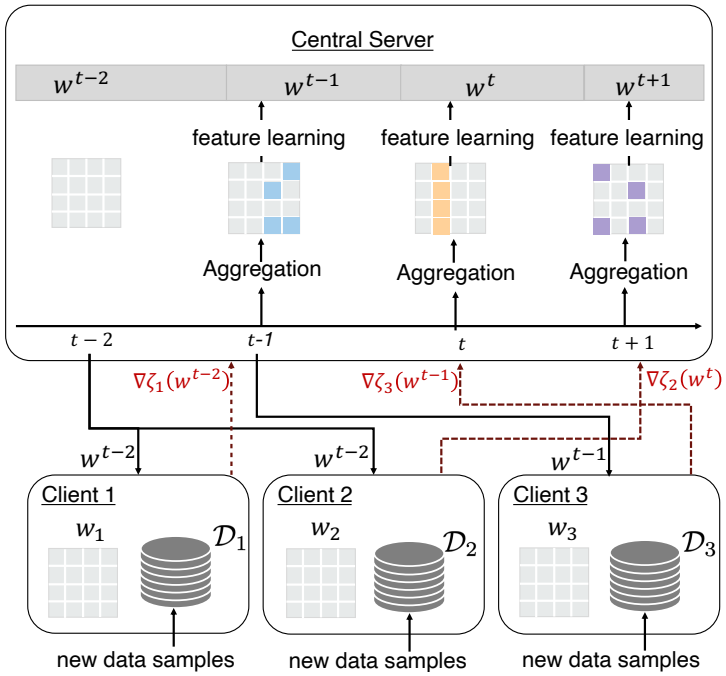


Figure 5.1: Illustration of update procedure for the proposed **ASO-Fed** model. Server aggregates after receiving update from one client, and local clients may have new data samples during the training process. Each w is used to represent the whole central/local model, $\nabla\zeta$ is the gradient of local client.

comes to obtaining model parameters from the server. Such inconsistency is common in the real-world settings and is caused by data and system heterogeneity, or network delay. We address this problem by learning a global feature representation on the server and using a dynamic learning step size for training local clients.

5.2.1 Learning on Central Server

The server aggregates the central model w after each global iteration. At global iteration $t+1$, assume the server receives an update from client k. Let w^{t+1} be the server model, w_k^t be the local model of client k at iteration t, $\nabla\zeta_k$ be the gradient on the local data of client k, η_k^t be the learning rate of client k and $N' = n_1 + \cdots + n_k' + \cdots + n_K$ be the current total number of data samples where n_k and N becomes n_k' and N' due to new data at client k.

By aggregating the update from client k, the server update is computed as follows:

$$\begin{aligned} w^{t+1} &= w^t - \frac{n'_k}{N'}(w_k^t - w_k^{t+1}) \\ &= w^t - \frac{n'_k}{N'}(w_k^t - (w_k^t - \eta_k^t \nabla \zeta_k(w^t))) \\ &= w^t - \eta_k^t \frac{n'_k}{N'} \nabla \zeta_k(w^t). \end{aligned} \tag{5.4}$$

Feature Representation Learning on Server. To address the potential effect on model performance caused by asynchronous updates, we apply feature representation learning on the server to extract a cross device feature representation. Attention mechanisms have shown to be effective in identifying key features and their representations [123, 124]. Our feature learning approach is inspired by this, and additionally, we combine weight normalization to reduce the computation cost [125, 126]. We use simple network architectures in this study so that it can be easily handled by mobile devices. We apply feature extraction on the first layer (e.g., LSTM or CNN in this chapter) after the input to generate the feature representation, and denote the parameters of this layer as $w_{(1)}^{t+1}$. For each element $w_{(1)}^{t+1}[i,j]$ in column $w_{(1)}^{t+1}[j]$ of $w_{(1)}^{t+1}$, we adopt the below operations to obtain the updated $w_{(1)}^{t+1}$:

$$\alpha_{(1)}^{t+1}[i,j] \leftarrow \frac{\exp(|w_{(1)}^{t+1}[i,j]|)}{\sum_j \exp(|w_{(1)}^{t+1}[i,j]|)}, \tag{5.5}$$

$$w_{(1)}^{t+1}[i,j] = \alpha_{(1)}^{t+1}[i,j] * w_{(1)}^{t+1}[i,j]. \tag{5.6}$$

5.2.2 Learning on Local Clients

In order to mitigate the deviations of the local models from the central model, instead of just minimizing the local function f_k, device k applies a gradient-based update using the

Algorithm 3 Algorithm for ASO-Fed

Input: Multiple related learning clients distributed at client devices, regularization parameter λ, multiplier r_k, learning rate $\bar{\eta}$, decay coefficient β.
Initialize: $h_k^{pre} = h_k = 0$, $v_k = 0$
Procedure at Central Server
for global iterations $t = 1, 2, ..., T$ **do**
    /* get the update on w^t */
    compute w^t ▷ [Eq.(6.5)]
    update w^t with feature learning ▷ [Eq.(5.5) - Eq.(5.6)]
end for
Procedure of Local Client k at round t
receive w^t from the server
Compute ∇s_k
Set $h_k^{(\text{pre})} = h_k$
Set $\nabla \zeta_k \leftarrow \nabla s_k - \nabla s_k^{(\text{pre})} + h_k^{(\text{pre})}$ ▷ [Eq.(5.7) -Eq.(5.10)]
Update $w_k^{t+1} \leftarrow w_k^t - r_k^t \bar{\eta} \nabla \zeta_k$
Compute and update $h_k = \beta h_k + (1 - \beta) v_k$
Update $v_k = \nabla s_k(w^t; w_k^t)$
upload w_k^{t+1} to the server

following surrogate objective s_k:

$$s_k(w_k) = f_k(w_k) + \frac{\lambda}{2}||w_k - w||^2. \tag{5.7}$$

Local Update with Decay Coefficient. Data continue arriving at local clients during the training process, so each client needs to perform online learning. For this process, each client requests the latest model from the server and updates the model with its new data. Thus there needs to be a balance between previous model and current model. At global iteration t, device k receives model w^t from the server. Let $\nabla s_k^{(\text{pre})}$ be the previous local gradients, the optimization of device k at this iteration is formulated as:

$$\nabla \zeta_k \leftarrow \nabla s_k - \nabla s_k^{(\text{pre})} + h_k^{(\text{pre})}, \tag{5.8}$$

$$h_k^{(\text{pre})} = \beta h_k^{(\text{pre})} + (1 - \beta) \nabla s_k^{(\text{pre})}. \tag{5.9}$$

where $h_k^{(\text{pre})}$ is used to balance the previous and current local gradients and initialized to be 0, β is the decay coefficient to balance the previous model and the current model. The update procedure of $h_k^{(\text{pre})}$ can be found in Algorithm 3.

With η_k^t being the learning rate for client k, the closed form solution for model update of client k is given by:

$$\begin{aligned} w_k^{t+1} &= w_k^t - \eta_k^t \nabla \zeta_k(w^t) \\ &= w_k^t - \eta_k^t \Big(\nabla f_k(w_k^t) - \nabla s_k^{(\text{pre})} + h_k^{(\text{pre})} + \lambda(w_k^t - w^t) \Big). \end{aligned} \tag{5.10}$$

Dynamic Learning Step Size. In real-world settings, the activation rates, i.e., how often clients provide updates to the central model, vary due to a host of reasons. Devices with low activation rates are referred as stragglers, which are caused by several reasons such as communication bandwidth, network delay or data heterogeneity. Thus, we apply a dynamic learning step size with the intuition that if a client has more data or poor communication bandwidth, the activation rate of this client towards the central update will be small and thus the corresponding learning step size should be large. Dynamic learning step sizes are used in asynchronous optimization to achieve better learning performance [127, 128]. Initially, we set $\eta_k^t = \bar{\eta}$ for all clients. The update process (5.10) can be revised as:

$$w_k^{t+1} = w_k^t - r_k^t \bar{\eta} \nabla \zeta_k(w^t). \tag{5.11}$$

where r_k^t is a time related multiplier, and is given by $r_k^t = \max\{1, \log(\bar{d}_k^t)\}$, where $\bar{d}_k^t = \frac{1}{t} \sum_{\tau=1}^{t} d_k^\tau$ is the average time cost of the past t iterations. Then the actual learning step size is scaled by the past communication delays. This dynamic learning step size strategy can reduce the effect of stragglers on model convergence. Since the stragglers usually have longer delays, the larger step sizes are assigned to these lagging clients to compensate for the loss.

5.2.3 Convergence Analysis

In this section, we prove theoretical analysis on the convergence of ASO-Fed. First, we introduce some definitions and assumptions for our convergence analysis.

Definition 3. (Smoothness) The function f has Lipschitz continuous gradients with constant $L > 0$ (in other words, f is *L-smooth*) if $\forall x_1, x_2$,

$$f(x_1) - f(x_2) \leq \langle \nabla f(x_2), x_1 - x_2 \rangle + \frac{L}{2}||x_1 - x_2||^2. \tag{5.12}$$

Definition 4. (Strong convexity) The function f is μ-*strongly convex* with $\mu > 0$ if $\forall x_1, x_2$,

$$f(x_1) - f(x_2) \geq \langle \nabla f(x_2), x_1 - x_2 \rangle + \frac{\mu}{2}||x_1 - x_2||^2. \tag{5.13}$$

In order to quantify the dissimilarity between devices in a federated network, following Li *et al* [129], we define the following definition on local non-IID data.

Definition 5. (Bounded gradient dissimilarity): The local functions ζ_k are V-locally dissimilar at w if $\mathbb{E}||(\nabla \zeta_k(w))||^2 \leq ||\nabla F(w)||^2 V^2$.

With Definition 5 we further define $V(w) = \sqrt{\frac{\mathbb{E}||\nabla F(w)||^2}{||(\nabla \zeta_k(w))||^2}}$ when $||(\nabla \zeta_k(w))||^2 \neq 0$. When all the local functions are the same, which is the samples on all the devices are in IID fashion, we have $V(w) \to 1$ for all w [129]. However, in federated setting with heterogeneous data, we often have $V > 1$ due to device discrepancies. Therefore, $V \geq 1$, and the larger V is, the larger the dissimilarity among the local functions, which is the more heterogeneous the local data.

Further, we make the following assumptions on the objective functions and introduce one lemma.

Assumption 1. *Suppose that:*

1. The central objective function $F(w)$ is bounded from below, i.e., $F_{\min} = F(w_*) > -\infty$.

2. There exists $\epsilon > 0$ *such that* $\mathbb{E}(\nabla\zeta_k(w)) \leq ||\nabla F(w)||$, *and* $\nabla F(w)^\top \mathbb{E}(\nabla\zeta_k(w)) \geq \epsilon||\nabla F(w)||^2$ *holds for all* w.

Note that if $\epsilon = 1$, then $\nabla\zeta_k(w)$ is an unbiased estimator of $\nabla F(w)$.

Lemma 1. *If* $F(w)$ *is* μ*-strongly convex, then with Assumption 1.1, we have:*

$$2\mu(F(w^t) - F(w_*)) \leq ||\nabla F(w^t)||^2. \tag{5.14}$$

While the proof of **Lemma 1** is supported by the literature [130, 131], we also provide a detailed proof in Section 5.6.

Theorem 1. *Suppose that the central objective function* $F(w)$ *is L-smooth and* μ*-strongly convex. Assume the local functions* ζ_k *are bounded dissimilar. Let Assumption 1 hold. Let* $\bar{\eta}_k \leq \eta_k^t < \eta_k = \frac{2\epsilon N'}{LV^2 n_k'}$, *then after* T *global updates on the server, ASO-Fed converges to a global optimum* w_**:*

$$\mathbb{E}(F(w^T) - F(w_*)) \leq (1 - 2\mu\gamma'\bar{\eta}_k)^T (F(w^0) - F(w_*)) \tag{5.15}$$

where $\gamma' = \epsilon - \frac{L\eta_k V^2}{2}$.

The detailed proof of Theorem 1 is provided in Section 5.6. Theorem 1 converges under the special case of convex central loss and gives an error bound for the general form of model aggregation.

Theorem 2. *Suppose that the central objective function* $F(w)$ *is L-smooth and non-convex. Let Assumption 1 hold. Assume the local functions* ζ_k *are bounded dissimilar. If it holds that* $\eta_k^t < \frac{2\epsilon - 1}{LV^2} \leq \max(r_k^t \bar{\eta}) = \bar{\eta}$ *for all* t, *then after* T *global iterations, we have*

$$\sum_{t=0}^{T-1} \frac{\eta_k^t}{2} \mathbb{E}(||\nabla F(w^t)||^2) \leq F(w^0) - F(w_*) \tag{5.16}$$

We direct the reader to Section 5.6 for a detailed proof of Theorem 2. The model convergence rate can be controlled with a balance between the bounded gradient dissimilarity value V and the learning rate η_k^t.

5.3 Experimental Setup

5.3.1 Comparative Methods

Baseline Methods

We compare the proposed ASO-Fed with the following synchronous and asynchronous federated learning approaches, single-client and global models.

- FedAvg [3–5]: the commonly used synchronous federated learning approach proposed by McMahan *et al.* [3].
- FedProx [129]: synchronous federated learning framework with a proximal term on the local objective function to mitigate the data heterogeneity problem and to improve the model stability compared to FedAvg.
- FedAsync [24]: asynchronous federated learning framework using a weighted average to update the server model.
- Local-S: each client learns separate model with its own data, and the model structure is the same as ASO-Fed.
- Global: combining the data of all clients and processed in a batch setting on a single machine.

Ablation Studies

We also perform ablation studies to study the effect of feature representation learning on server and local dynamic learning step size:

- ASO-Fed(-D): the proposed ASO-Fed without dynamic learning step size.

- ASO-Fed(-F): the proposed ASO-Fed without central feature representation learning.

5.3.2 Training Details

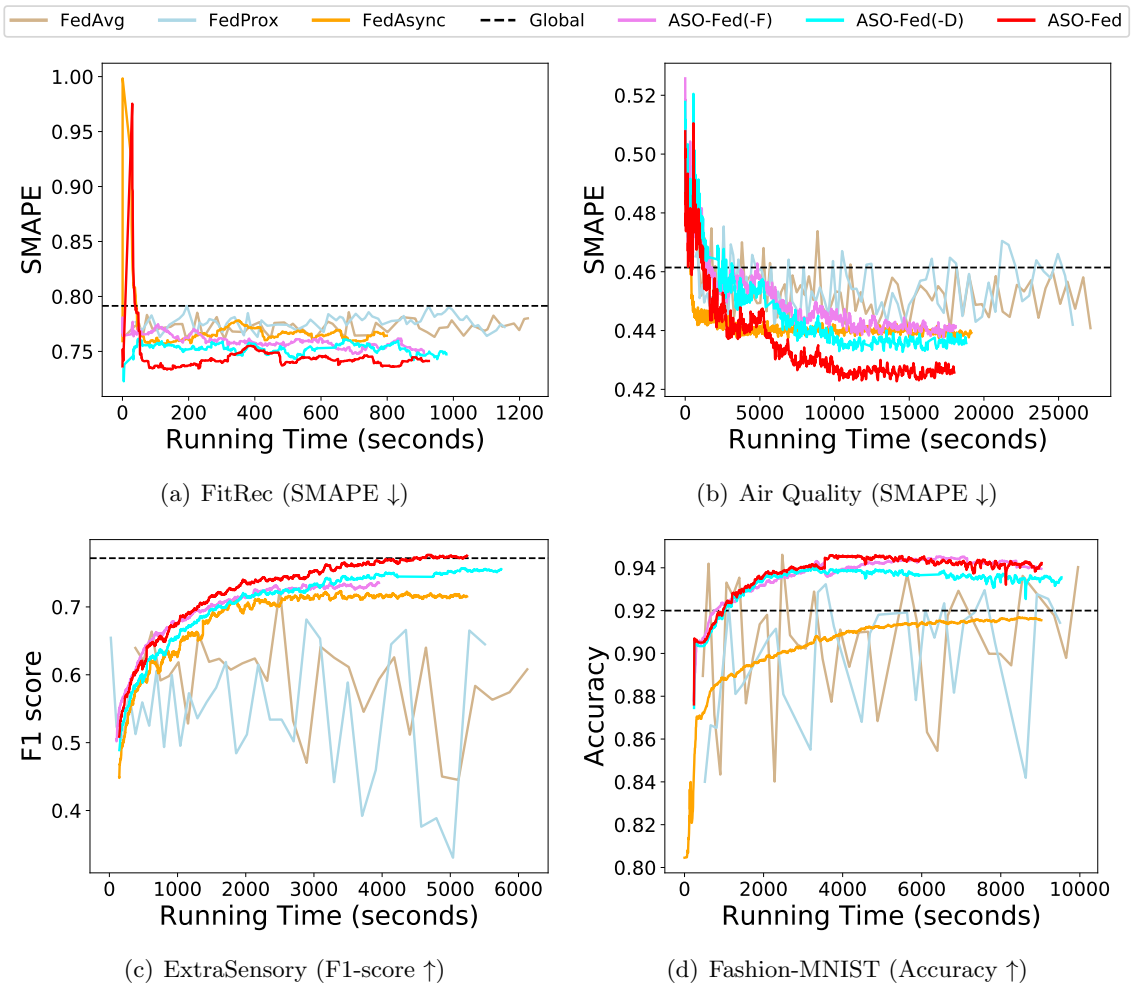
For each dataset, we split the each client's data into 60%, 20%, 20% for training, validation, and testing, respectively. As for each client's training data, we start with a random portion of the total training size, and increase by $0.05\%-0.1\%$ each iteration to simulate the arriving data. We set the fraction C of FedAvg as 0.2, decay coefficient β as 0.001, $\bar{\eta} = 0.001$, $\lambda = 1.0$ for FitRec and Air Quality datasets, $\lambda = 0.8$ for ExtraSensory dataset, and $\lambda = 0.5$ for Fashion-MNIST. For FedAsync model, we set $\gamma = 0.1$, $\rho = 0.005$ and $\alpha = 0.6$. We use a single layer LSTM followed by a fully connected layer for the three real-world datasets and two CNN layers followed by a max pooling layer for Fashion-MNIST. The local epoch number of each client is set as 2. We design simple network architectures for all datesets so that it can be easily handled by mobile devices. All of the experiments are conducted with two Intel E5-2660 v3 10-core CPU at 2.60GHz [132].

Simulation parameters. To simulate the stragglers and dropouts situations, we set different network settings for our experiments, a random offset parameter ($10 \sim 100$ seconds) was taken as an input from the client. This parameter represents the average delay related to the infrastructure of the network for a client. We direct the reader to Section 5.4 for the detailed results.

5.4 Experimental Results

5.4.1 Predictive Performance Comparison

Table 7.1 reports the predictive performance comparing ASO-Fed to the baseline approaches. In case of regression problems, we report the average *MAE* and *SMAPE* values, for ExtraSensory classification benchmark we report the average *F1*, *Precision*, *Recall* and *Balanced Accuracy (BA)*, and for Fashion-MNIST we report the *Accuracy*. From Table 7.1, we



(a) FitRec (SMAPE ↓)

(b) Air Quality (SMAPE ↓)

(c) ExtraSensory (F1-score ↑)

(d) Fashion-MNIST (Accuracy ↑)

Figure 5.2: Test set performance vs. running time for four datasets. Lower SMAPE value indicates better model performance. For the synchronized federated frameworks, we plot results of *FedAvg* and *FedProx* at every 10 global iterations.

Table 5.1: Prediction performance comparison. Bold numbers are the best performance, numbers with underlines are the second best values. improv.(1) shows the percentage improvement of ASO-Fed over FedAvg. improv.(2) shows the percentage improvement of ASO-Fed over the best baseline results.

Method	**FitRec**				**Air Quality**		**ExtraSensory**				**Fashion-MNIST**
	MAE ↓ (Speed)	SMAPE↓ (Speed)	MAE↓ (HeartRate)	SMAPE↓ (HeartRate)	MAE↓	SMAPE↓	F1↑	Precision↑	Recall↑	BA↑	Accuracy↑
FedAvg	13.61	0.78	13.72	0.78	44.30	0.44	0.66	0.87	0.55	0.77	0.87
FedProx	14.21	0.82	14.53	0.83	44.30	0.44	0.67	0.82	0.57	0.77	0.88
FedAsync	13.56	0.78	13.67	0.78	37.98	0.43	0.72	0.84	0.65	0.82	0.90
Local-S	12.76	0.75	13.27	0.76	36.72	0.56	0.65	0.72	0.61	0.79	0.89
Global	12.95	0.78	12.79	0.79	37.61	0.44	**0.77**	**0.92**	0.66	0.83	0.92
ASO-Fed(-D)	12.46	0.74	12.51	0.75	37.13	0.43	0.76	0.88	0.69	**0.85**	0.94
ASO-Fed(-F)	12.62	0.76	12.71	0.76	37.72	0.43	0.75	0.86	0.68	0.84	0.94
ASO-Fed	**12.31**	**0.73**	**12.36**	**0.74**	**36.71**	**0.42**	**0.77**	0.88	**0.70**	**0.85**	**0.95**
improv.(1)	9.55%	6.41%	9.91%	5.13%	17.13%	2.32%	16.66%	1.15%	27.27%	10.39%	9.19%
improv.(2)	3.52%	2.67%	3.36%	2.63%	0.03%	4.54%	0.00%	-4.34%	6.06%	2.41%	3.26%

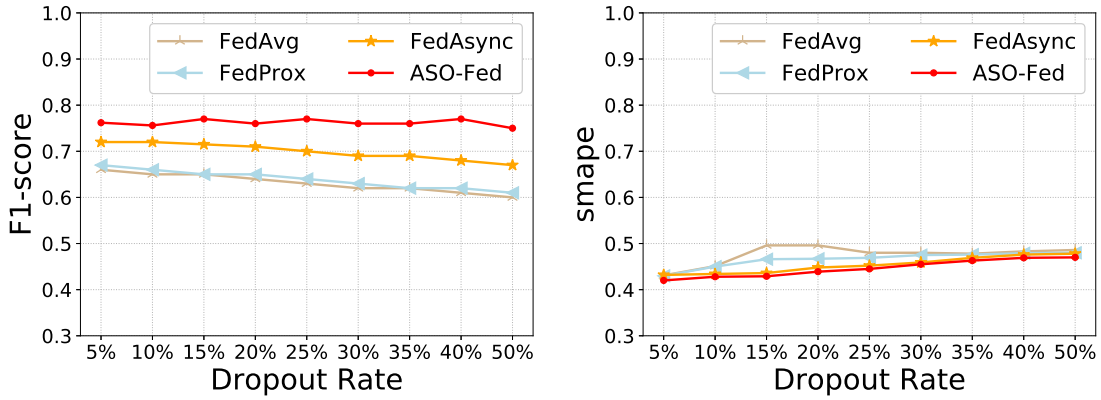
observe that ASO-Fed achieves the lowest MAE and SMAPE values for FitRec and Air Quality datasets, and has the overall best performance for ExtraSensory and Fashion-MNIST datasets. Across the four datasets, ASO-Fed outperforms the Global model (acquires all data at a single server) by $2.39\% \sim 6.41\%$. FedAvg and FedProx do not perform well on the highly unbalanced and non-IID datasets (FitRec, ExtraSensory and Fashion-MNIST). The interpretation is that, with non-IID and streaming data, local models will diverge further from the server model, while the synchronous FL models cannot balance the previous local model and the current local model, thus fail to mitigate the gap between local and server models. In particular, for the FitRec and Fashion-MNIST datasets, Local-S (the local single client model) outperforms FedAvg and FedProx.

Figure 6.2 shows the prediction performance for the different approaches as a function of running time. From Figure 6.2, we notice large fluctuations on the performance of FedAvg and FedProx during the whole training process on all four benchmarks. This shows that synchronous federated frameworks do not perform well on streaming data with skewed and non-IID data distribution. FedAsync achieves better performance than the two synchronous federated frameworks, but not as good as ASO-Fed. ASO-Fed has steady improvements with running time (and converges quickly).

Ablation Studies. To evaluate the performance of central feature representation learning

Table 5.2: Computation time (in minutes) to reach target test performance. The network delay of each client was set to be a random value between $10 \sim 100$ seconds.

Method	**FitRec**	**Air Quality**	**ExtraSensory**	**FMNIST**
FedAvg	20.42	460.02	104.86	160.72
FedProx	19.26	439.95	99.45	160.36
FedAsync	15.41	326.45	87.97	151.72
ASO-Fed(-D)	16.31	332.74	95.77	158.83
ASO-Fed(-F)	**15.17**	320.92	**65.87**	150.54
ASO-Fed	15.43	**319.41**	87.40	**150.46**

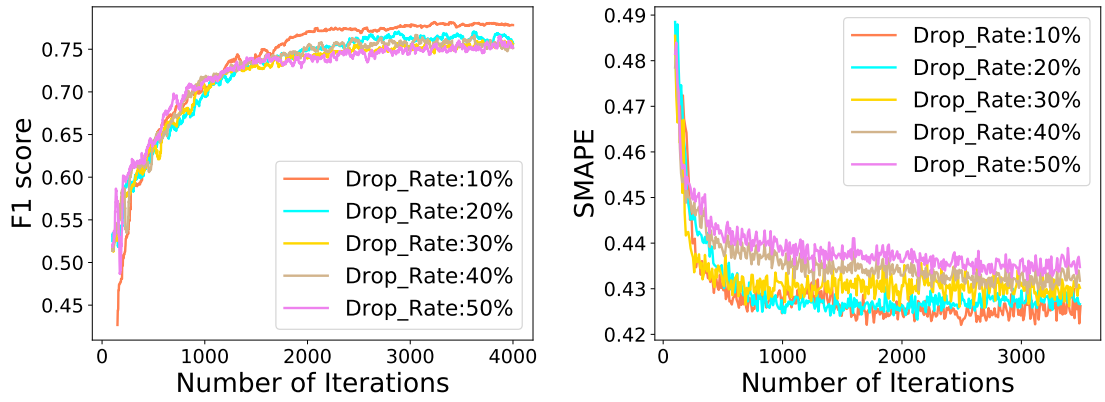


(a) ExtraSensory (F1-socre ↑)

(b) Air Quality (SMAPE ↓)

Figure 5.3: Performance comparison of federated approaches as dropout rate of clients increases. ASO-Fed has better performance than the other federated frameworks.

and local dynamic learning step size, we perform ablation studies with two additional models: ASO-Fed(-F) and ASO-Fed(-D). From Table 7.1 we notice that ASO-Fed outperforms ASO-Fed(-F) by $1.06\% \sim 5.26\%$, which shows the effectiveness of central feature learning at generating a better feature representation across clients. We show the visualizations of learned features for the three real-world datasets in Section 5.4.5. ASO-Fed(-D) has close prediction performance as ASO-Fed, but as shown in Figure 6.2, ASO-Fed(-D) needs a longer training time to converge than ASO-Fed. This shows that local dynamic learning approach works effectively at lowering the overall computation cost.



(a) ExtraSensory (F1-score ↑)

(b) Air Quality (SMAPE ↓)

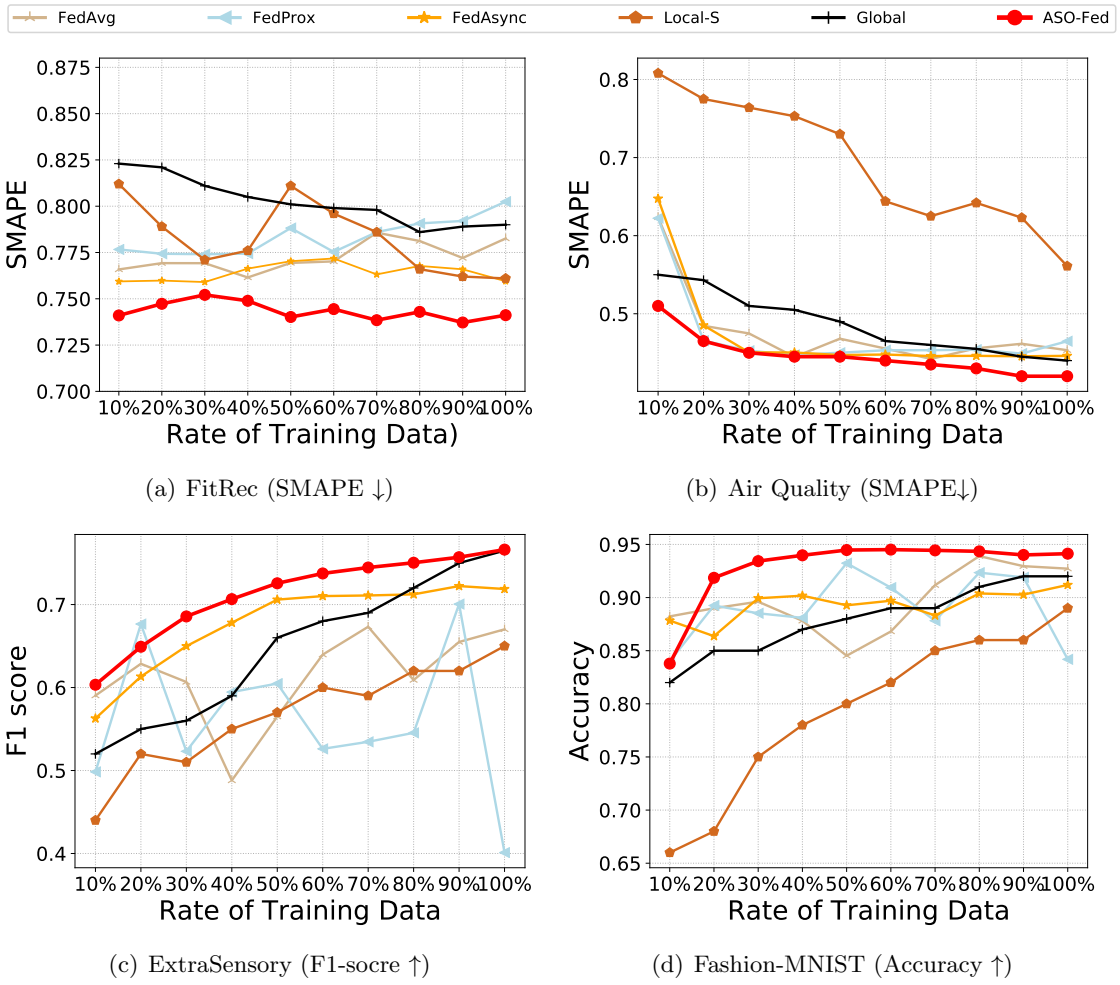
Figure 5.4: The performance of ASO-Fed with clients periodically dropping out.

5.4.2 Evaluation of Time Efficiency

We report the running time of synchronous and asynchronous FL approaches to reach target test performance in Table 5.2. As seen from this table, FedAvg and FedProx have the highest computation cost across all four benchmarks. This is expected given that in synchronous protocol, the server aggregation has to wait for the slow client nodes to finish their computations. Among the comparison of asynchronous update models, ASO-Fed is more time efficient than ASO-Fed(-D). FedAsync is close in running time as ASO-Fed(-D). From the empirical results we note that the dynamic learning step size is a promising strategy and effective when there are significant delays in the network. ASO-Fed is close to ASO-Fed(-F) in terms of computational cost except for the FitRec and ExtraSensory dataset. Because these two datasets have more complex features and additional computation for feature extraction requires more time, but ASO-Fed obtains better prediction performance with a little sacrifice of time efficiency.

5.4.3 Robustness to Stragglers and Dropouts

Stragglers are clients that lag in providing updates to the server due to a variety of reasons: communication bandwidth, computation load, and data variability. In this Section, we



(a) FitRec (SMAPE ↓)

(b) Air Quality (SMAPE↓)

(c) ExtraSensory (F1-socre ↑)

(d) Fashion-MNIST (Accuracy ↑)

Figure 5.5: Average performance comparison (SMAPE, F1, accuracy) on four datasets as training data increases.

investigate a common real-world scenario, when clients have no response during the entire training process or are not available in certain time windows. We refer to these non-contributing clients as dropouts.

We set a certain portion of clients to be non-responsive. These clients will not participate in the training process. However, the reported results are evaluated on test data from all clients. Figure 5.3 shows the performance of federated learning approaches on Extrasensory classification and Air Quality regression benchmarks with an increasing fraction of clients being dropped from the learning process. As shown in Figure 5.3, for the ExtraSensory

dataset, we observe that as the rate of dropout clients increases, the performance of the FedAvg model also drops. The same trend is noticed for FedProx. As for ASO-Fed, the prediction performance is steady except for a slight decrease when the dropout rate exceeds 40%. Even when 50% of clients are subject to dropout during training, ASO-Fed can still achieve at least a 10% improvement over synchronous FL models and FedAsync. For the Air Quality dataset, ASO-Fed has lower SMAPE errors than all other models as the dropout rate increases and the performance of ASO-Fed is relatively stable. However, as expected, if one of the nodes never sends updates to the central server, the model does not generalize. This explains the poor performance as the dropout rate increases.

We also explore the performance effect of nodes periodically dropping and not providing updates to the server. To the best of our knowledge, no other methods for asynchronous federated learning with local streaming data directly address this issue. Therefore we do not draw comparisons to other methods. At each global iteration, we randomly select a certain fraction of clients as not participating in the training during the current iteration. Figure 5.4 shows the convergence trend for different rates of periodically dropping clients. We notice that as the rate of periodically dropout clients increases, ASO-Fed still converges well with slightly worse performance. This shows that the performance of ASO-Fed is robust to relatively high rates of periodical dropouts.

5.4.4 Results of Varying Training Samples

To evaluate the incremental online learning process more explicitly, we display how the prediction performance changes with an increasing number of training samples in Figure 5.5. We perform experiments with different rates of all clients' training data and depict the average performance on all clients. From Figure 5.5, for all datasets, ASO-Fed achieves the best performance with increasing rates of training data. Large fluctuations are observed in the results of FedAvg and FedProx, which exhibit an unstable model performance for synchronous approaches as the local data sets increase. Compared with the Global approach, asynchronous frameworks have more stable performance as the amount of training data

increases, while the individual models learned by Local-S do not perform well. The analysis shows that that ASO-Fed learns an effective model with a smaller portion of training data. With increasing local data, ASO-Fed still outperforms the other competitors.

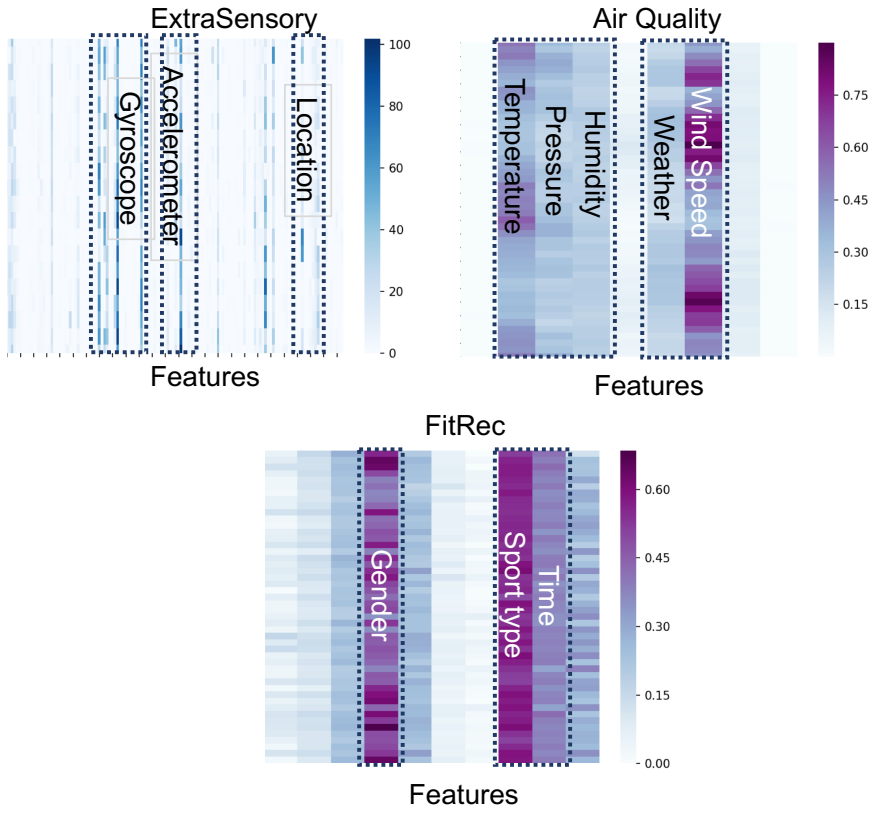


Figure 5.6: Feature representation learned on the server of three real-world datasets. Each column is the weights vector within 48 time steps over the input series.

5.4.5 Feature Representation learning

In this section, we present the qualitative results of the proposed feature representation learning on the server. In Figure 5.6, we show the features learned from one client of three datasets respectively. For the client in ExtraSensory, the highlighted features are 'Gyroscope', 'Accelerometer' and 'Location', and the corresponding labels are 'walking', 'at_home'. For the client from Air Quality dataset, we observe that features with high weights are 'Wind Speed' and 'Temperature'. This makes sense given that the target values

are air pollutants (e,g,. PM2.5, SO) and 'Wind Speed' decides whether these pollutants can be dispersed. Air pollutants vary with seasons, and a higher concentration of air pollutants appears in winter time due to fuel consumption for heating in winter. Therefore 'Temperature' is also a strong indicator for air pollutants. For the client from FitRec, the extracted features are 'gender', 'sport type' and 'time'. Since the prediction targets are speed and heart rate, these three features have strong correlations with the targets. The above results show the effectiveness of feature learning in ASO-Fed.

5.5 Summary

In this study, I propose a novel asynchronous online federated learning approach to tackle the learning problems on distributed edge devices. To the best of my knowledge, this is the first attempt to solve asynchronous federated learning with local streaming data. Compared to synchronized FL approaches (FedAvg and FedProx), ASO-Fed is computationally efficient since the central server does not need to wait for lagging clients to perform aggregation. Compared with asynchronous approach (FedAsync), the proposed approach achieves better performance on all provided datasets, which indicates that the proposed asynchronous update method can better handle local streaming data. Time efficiency is compared on multiple benchmarks and the results show that the proposed ASO-Fed is faster than synchronized FL. The proposed ASO-Fed inherits the idea of using asynchronous update scheme as [24, 25]. Likewise, it shares the same communication bottleneck problem pointed out in [19]. Nevertheless, it is still an open issue to build communication efficient methods in asynchronous federated learning frameworks. Besides, feature learning component needs longer computation time when dealing with datasets which have complex features. Thus, there is a trade off between the prediction performance and computational costs.

5.6 Convergence Analysis

5.6.1 Proof of Lemma 1

Proof. $F(w)$ is *μ-strongly convex*, we can get:

$$F(w') - F(w^t) \geq \langle \nabla F(w^t), w' - w^t \rangle + \frac{\mu}{2}||w' - w^t||^2, \tag{5.17}$$

Let us define $C(w')$ such that:

$$C(w') = F(w^t) + \langle \nabla F(w^t), w' - w^t \rangle + \frac{\mu}{2}||w' - w^t||^2, \tag{5.18}$$

$C(w')$ is a quadratic function of w', then it has minimal value when $\nabla C(w') = \nabla F(w^t) + \mu(w' - w^t) = 0$. Then the minimal value of $C(w')$ is obtained when $w' = w^t - \frac{\nabla F(w^t)}{\mu}$, which is:

$$C_{\min} = F(w^t) - \frac{||\nabla F(w^t)||^2}{2\mu}, \tag{5.19}$$

For $F(w)$ is *μ-strongly convex*, we can complete the proof:

$$F(w_*) \geq C(w_*) \geq C_{\min} = F(w^t) - \frac{||\nabla F(w^t)||^2}{2\mu}, \tag{5.20}$$

$$2\mu(F(w^t) - F(w_*)) \leq ||\nabla F(w^t)||^2. \tag{5.21}$$

□

5.6.2 Proof of Theorem 1

Proof. With $F(w)$ is *L-smooth*, we have:

$$\begin{aligned} F(w^{t+1}) - F(w^t) &\leq \langle \nabla F(w^t), w^{t+1} - w^t \rangle + \frac{L}{2}||w^{t+1} - w^t||^2 \\ &= -\nabla F(w^t)^\top \eta_k^t \frac{n'_k}{N'} \nabla \zeta_k(w^t) + \frac{L}{2}||\eta_k^t \frac{n'_k}{N'} \nabla \zeta_k(w^t)||^2, \end{aligned} \tag{5.22}$$

let $m = \eta_k^t \frac{n'_k}{N'} > 0$, with Assumption 1 and local Bounded gradient dissimilarity, we can get:

$$\begin{aligned} &\mathbb{E}(F(w^{t+1})) - F(w^t) \\ &\leq -m \nabla F(w^t)^\top \mathbb{E}(\nabla \zeta_k(w^t)) + \frac{Lm^2}{2} \mathbb{E}(||\nabla \zeta_k(w^t)||^2) \\ &\leq -m\epsilon ||\nabla F(w^t)||^2 + \frac{Lm^2 V^2}{2} ||\nabla F(w^t)||^2 \\ &= -m(\epsilon - \frac{LmV^2}{2}) ||\nabla F(w^t)||^2, \end{aligned} \tag{5.23}$$

We can easily prove that $-m(\epsilon - \frac{LmV^2}{2})$ is monotonically increasing while $m > 0$. Since $n'_k < N'$, thus $m = \eta_k^t \frac{n'_k}{N'} < \eta_k^t$. Then we have $-m(\epsilon - \frac{LmV^2}{2}) < -\eta_k^t(\epsilon - \frac{L\eta_k^t V^2}{2})$.

With Lemma 1, and let $\gamma = \epsilon - \frac{L\eta_k^t V^2}{2}$, we can rewrite Equation (5.23) as:

$$\mathbb{E}(F(w^{t+1})) - F(w^t) \leq -2\mu\eta_k^t \gamma (F(w^t) - F(w_*)), \tag{5.24}$$

Then we move $F(w^t)$ on left side to right and subtract $F(w_*)$ from both sides, and get:

$$\mathbb{E}(F(w^{t+1})) - F(w_*) \leq (1 - 2\mu\eta_k^t \gamma)(F(w^t) - F(w_*)), \tag{5.25}$$

Since $\bar{\eta}_k < \eta_k^t$, then by taking expectation of both sides, and telescoping, we have:

$$\mathbb{E}(F(w^{t+1}) - F(w_*)) \leq (1 - 2\mu\bar{\eta}_k\gamma')(\mathbb{E}(F(w^t) - F(w_*)). \tag{5.26}$$

When $t + 1 = T$, the above inequality becomes Equation (5.15). Thus we complete the proof. □

5.6.3 Proof of Theorem 2

Proof. With $m < \eta_k^t$ and $\gamma = \epsilon - \frac{L\eta_k^t V^2}{2}$, we replace m with η_k^t and take the full expectation of Equation (5.23), we have:

$$\mathbb{E}(F(w^{t+1})) \leq \mathbb{E}(F(w^t)) - \eta_k^t\gamma\mathbb{E}(||\nabla F(w^t)||^2), \tag{5.27}$$

Then summing up (5.27) over global iteration T, we can get:

$$\mathbb{E}(F(w^{t+1})) \leq F(w^0) - \sum_{t=0}^{T-1} \eta_k^t\gamma\mathbb{E}(||\nabla F(w^t)||^2), \tag{5.28}$$

From Assumption 1.1 we can get $F(w_*) \leq \mathbb{E}(F(w^{t+1}))$, then we have:

$$F(w_*) \leq F(w^0) - \sum_{t=0}^{T-1} \eta_k^t\gamma\mathbb{E}(||\nabla F(w^t)||^2), \tag{5.29}$$

If we set $\eta_k^t < \frac{2\epsilon-1}{LV^2} \leq max(r_k^t\bar{\eta}) = \bar{\eta}$, we can get $\eta_k^t(\epsilon - \frac{L\eta_k^t V^2}{2}) > \frac{\eta_k^t}{2}$. Rearrange Equation (5.29) we can get:

$$\begin{aligned}\sum_{t=0}^{T-1} \frac{\eta_k^t}{2}\mathbb{E}(||\nabla F(w^t)||^2) &\leq \sum_{t=0}^{T-1} \eta_k^t(\epsilon - \frac{L\eta_k^t V^2}{2})\mathbb{E}(||\nabla F(w^t)||^2) \\ &\leq F(w^0) - F(w_*).\end{aligned} \tag{5.30}$$

Then we get Equation (5.16) and complete the proof. □

Chapter 6: Asynchronous Federated Learning for Sensor Data with Concept Drift

Previous work on asynchronous federated learning propose an online learning approach to handle the streaming device data [133]. While this approach is novel at tackling the challenges associated with heterogeneous edge devices, it cannot deal with the concept drift on local device data efficiently. Inspired by this work, I further propose a novel asynchronous federated learning method, FedConD, which can detect and handle concept drift on local devices adaptively.

In addition, although [133] provides dynamic learning strategy on local devices to deal with the vary latency among devices, the central model may still bias to the devices with lower latency as they communicate more updates to the server. Allowing the server to broadcast the central model to all devices at each round is also a waste of communication bandwidth. To deal with the above mentioned problems, I design a new strategy which allows the server to send update request to devices with high latency (e.g., devices with low bandwidth or poor hardware) instead of sending to all devices blindly. This will balance the update frequency among devices and lead to a more optimal central model. Meanwhile, the overall communication between the server and devices is also reduced with a wise selection of local updates. The work presented in this chapter has been submitted to a conference and under review.

6.1 Problem Description

In this section, we define the classical federated learning objective and introduce the definition of concept drift.

6.1.1 Preliminaries: Classical Federated Learning

For classical federated learning algorithms, the goal is to minimize the following objective function:

$$\min_{w} \left\{ F(w) = \sum_{k=1}^{K} p_k f_k(w). \right\} \tag{6.1}$$

where K is the total number of devices, $p_k \geq 0$ is the weight of k-th device and $\sum_{k=1}^{K} p_k = 1$. Suppose $\mathcal{D}_k$ is data captured on device k, and let $n_k = |\mathcal{D}_k|$ be the number of samples on device k. We can set p_k to be $\frac{n_k}{N}$, where $N = \sum_k n_k$ is the total number of samples over the entire dataset. The local objective of client k is defined as:

$$f_k(w_k) \stackrel{def}{=} \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \ell_i(x_i, y_i; w_k). \tag{6.2}$$

where $\ell_i(x_i, y_i; w_k)$ is the corresponding loss function[1] for data sample $\{x_i, y_i\}$ and w_k is the local model parameter.

6.1.2 Concept Drift

The distribution $\mathcal{D}_k$ may change over time. For a given time period $[0, t]$, a set of samples $S_{0,t} = (d_0, ..., d_t)$ are from $\mathcal{D}_k$, where $d_t = (x_t, y_t)$ is the data sample at time step t, x_t is the feature vector, y_t is the target label. Assume each training example on device k is generated by a source $\mathcal{S}_k$, then a sudden drift occurs when $\mathcal{S}_k^t$ is suddenly replaced by $\mathcal{S}_k^{t+1}$ at time step t. Gradual drifts change with a slower rate and roughly can be divided into two types. The first type has a mixed distribution of $\mathcal{D}_k$ and $\mathcal{D}_k'$ during the transition phase. As time goes on, the probability of observing examples from $\mathcal{D}_k$ decreases, while that of examples from $\mathcal{D}_k'$ increases. The second type also be referred as incremental drift [28],

[1] Cross-entropy loss for the classification problems and mean absolute error for the regression problems.

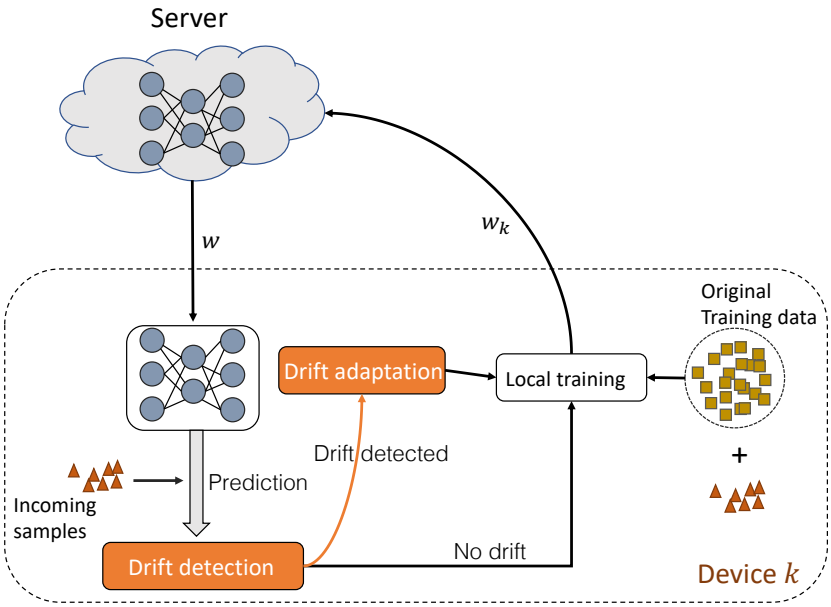


Figure 6.1: Illustration of the learning process on device k. After receiving the global model from server, local device first performs prediction on the new coming samples, then detecting whether there is concept drift. If drift happens, a drift adaptation strategy will be applied before the local training. w is used to represent the central model, and w_k is the trained model of device k.

which has more than two data distributions, however the difference between any adjacent distributions is small and the transition usually takes a longer period of time.

Concept drift is critical to online learning problems, because such inequality may lead to a inconsistency in decision boundaries, thereby increasing the error rate. Research into concept drift adaptation focuses on how to minimize the drop in accuracy and achieve the fastest recovery rate during the concept transformation process. We consider of both of these concept drift types and aim to minimize the performance drops of the learned models in this paper.

6.2 Method

In this section, we first introduce our strategy for concept drift detection and method to handle the concept drift on local devices. Then we explain the learning procedure on server.

To begin with, we first update the local objective function of client k:

$$h_k(w_k) = f_k(w_k) + \frac{\lambda}{2}||w_k - w||^2 \tag{6.3}$$

where w is the global model on the server. The regularization component $\frac{\lambda}{2}||w_k - w||^2$ aims to restrict the amount of local deviation by penalizing large changes from the current model at the server [133, 134]. By restricting the local updates to be closer to the initial (global) model, we can limit the impact of variable local updates efficiently.

Figure 6.1 illustrates the learning process on local device. The device starts to do prediction on the new incoming data after receiving the global model w from server. Next a drift detection process will be applied on the prediction results to determine whether a concept drift happens (detail explanation is in Section 6.2.1). If a drift is detected, the drift adaptation process is initialized before the training starts, otherwise the training will start directly. Finally the trained local model w_k will be uploaded to server for aggregation.

6.2.1 Local Drift Detection

The main idea behind our concept drift detection algorithm is that we can use the new generated samples on the devices for evaluating the models before we use them for training. Then using the results got from this evaluation we can decide whether the concept drifted.

For device k, at time step t the classifier predicts the class label of x_{t+1} to be $\hat{y}_{t+1}$. Most of previous works assume that the true label y_{t+1} is provided after some time, and both the y_{t+1} and $\hat{y}_{t+1}$ are available for the drift detection[**?**, 97, 112, 113], which is, the learning is in a supervised framework. Then, example x_{t+1} with its label y_{t+1} becomes a part of the training data and the process is repeated when the next instance is observed.

We perform drift detection using statistical testing. Each time we do prediction on the new incoming data with the downloaded server model, and then perform evaluation on the predicted labels and the true labels. Next we add the evaluation result into a queue with bounded size. Suppose device k has performed local training a times, and let s_k be the

Algorithm 4 Algorithm of FedConD

Input: K distributed devices with related data distribution, regularization parameter λ, threshold δ, parameter γ.
Initialize: a portion of γ devices to be available for training.
Learning at Server
for rounds $t = 1, 2, ..., T$ **do**
    /* get the update on w^t */
    compute w^t ▷ [Eq.(6.5)]
    broadcast w^t to the device with the fewest updates
end for
Learning Process of Local Client k at round t
receive w^t from the server
$s_k^{a+1} \leftarrow$ evaluate on new incoming samples
$\bar{s_k} = mean[s_k^1, s_k^2, ..., s_k^a]$
$\hat{s_k} = mean[s_k^1, s_k^2, ..., s_k^a, s_k^{a+1}]$
Perform statistical testing and get Γ_k, then use Γ_k to get the corresponding p-value P ▷ [Eq.(6.4)]
if $P > significance\,level$ **then**
    continue
else
    increase λ
Update $w_k^{t+1} \leftarrow w_k^t - \eta_k^t \nabla h_k$
upload w_k^{t+1} to the server

evaluation value, then the evaluation queue is $[s_k^1, s_k^2, ..., s_k^a]$. By storing these evaluations in the history, we can detect the trend of the performance of the model and decide whether there is a drift occurs at time $a+1$. We assume that s_k^{a+1} equals to $\bar{s}_k = \mu(s_k^1, s_k^2, ..., s_k^a)$ if no concept drift happens; and a significant decrease of s_k^{a+1} suggests that the concept is changing. The test is performed by calculating the following statistic[2]:

$$\Gamma_k = \frac{|\bar{s}_k - s_k^{\alpha+1}| - 0.5\Delta_k}{\sqrt{\hat{s}(1-\hat{s})\Delta_k}} \tag{6.4}$$

where $\Delta_k = \frac{1}{a} + 1$ and $\hat{s} = \mu(s_k^1, s_k^2, ..., s_k^a, s_k^{a+1})$. We compare its value to the percentile of the standard normal distribution to obtain the observed significance level (p-value). If

[2]For computational efficiency, we use the Yates's correction for the Pearson chi-square test.

the p-value, P, is less than a significance level, then the null hypothesis ($s_k^{a+1} = \bar{s}_k$) is rejected and the alternative hypothesis ($s_k^{a+1} > \bar{s}_k$) is accepted, namely concept drift has been detected. We set the significance level as 0.05 (5%), which can indicate a difference is significant if P is less than this value. The bounded size of the queue is set to be 20 for our cases.

6.2.2 Local Drift Adaptation

The added regularization component in Equation (6.3) is a penalty term to the loss function. For instance, Zhang el al. [37] propose elastic averaging SGD in deep networks with a similar penalty term in its objective. In distributed methods, Shamir et al. propose DANE [135] and Reddi et al. provide AIDE [78] use similar regularization term in the local objective function, while these methods are within the data center setting. Li el al. propose FedProx [134] to add this penalty term to tackle heterogeneity in federated networks, and they show that FedProx is robust to systems heterogeneity with stable convergence compared to vanilla FedAvg.

In Equation (6.3), λ controls the amount of penalty added to the original local objective $f_k(\cdot)$. When a drift is detected, the local model will deviate further from the global model, thus we should increase the weight of penalty component to force the local model to be close to the global model. We show empirically that this strategy can safely incorporate variable amounts of local work resulting from concept drift.

6.2.3 Learning on Server

The server aggregates the received local models in an asynchronous manner, that is, server model will be updated after receiving one device's update. Suppose at round $t+1$, server receives the update from device k. Let w^{t+1} be the server model, w_k^t be the local model, n_k^{t+1} be the number of samples on devices k and N^{t+1} be the total samples over all devices.

The server model is updated as follow:

$$
\begin{aligned}
w^{t+1} &= w^t - \frac{n_k^{t+1}}{N^{t+1}}(w_k^t - w_k^{t+1}) \\
&= w^t - \frac{n_k^{t+1}}{N^{t+1}}(w_k^t - (w_k^t - \eta_k^t \nabla h_k(w^t))) \\
&= w^t - \eta_k^t \frac{n_k^{t+1}}{N^{t+1}} \nabla h_k(w^t).
\end{aligned} \tag{6.5}
$$

where η_k is the learning rate of device k, and $\nabla h_k(\cdot)$ is the gradient on the local data of device k.

6.2.4 Communication-efficient Learning

Previous asynchronous FLs have no constrains on local device selection, that is, the server broadcasts global model w to all available devices at each round [24, 133]. This communication strategy is not efficient due to 1) downlink communication cost is high due to the server need to communicate with all local devices at each round; 2) the server can easily become a communication bottleneck with tens of thousands of devices updating the model simultaneously; 3) the update frequency of devices vary due to reasons like system heterogeneity, network bandwidth or data heterogeneity. Therefore, the aggregated global model may bias to devices which have more frequent updates.

In former work of communication-efficient federated learning, one commonly used approach to reduce the overall communication cost is compressing the model size transferred between the server and devices [86, 87, 91]. While these compression techniques can reduce the communication cost efficiently, the model performance is also hurt because the whole model information cannot be preserved during the compression-decompression process. Different from the synchronous FL frameworks, which allow only a portion of devices to perform local training and communicate with the server. With large amount of local devices, asynchronous FL framework should select local device update more wisely and

also control the number of local devices which perform training simultaneously. To address the above mentioned problems we propose a communication-efficient strategy to reduce the overall communication cost during the learning process in asynchronous FL frameworks, and furthermore, learn a more fair global model with faster convergence speed. At the server side we maintain records of the update frequency of all devices $\{p_1, p_2, ..., p_K\}$. At the beginning of each round, the server will send update request to the device with fewest updates, which is, getting $min(p_1, p_2, ..., p_K)$ and send server model to the according device. Meanwhile, we design a parameter γ to control the number of local devices which are performing local training at the same time. A detailed analysis on this parameter can be found in Section 6.4.3.

6.3 Experimental Evaluation

We perform extensive evaluations on three evolving data streams and two image datasets.

6.3.1 Datasets

- **FitRec Dataset**[3]**:** User sport records generated on mobile devices and uploaded to Endomondo, including multiple sources of sequential sensor data such as heart rate, speed, GPS as well as the sport type (e.g., biking, hiking), user gender and weather condition. Following [136], we re-sampled the data in 10-second intervals. We use data of randomly selected 30 users for heart rate prediction.

- **Air Quality Dataset**[4]**:** Air quality data collected from multiple weather sensor devices distributed in 9 locations of Beijing from Jan 2017 to Jan 2018, with features such as thermometer and barometer. Each area is modeled as a separate participant and the observed weather data is used to predict the measure of six air pollutants (e.g., PM2.5, PM10) from May 1st, 2018 to May 31st, 2018.

[3]https://sites.google.com/eng.ucsd.edu/fitrec-project/home
[4]https://biendata.com/competition/kdd_2018/data/

- **ExtraSensory Dataset [5]:** Mobile phone sensor data (e.g., location services, audio, accelerator) collected from 60 users, performing any of 51 activities (e.g., walking, talking, running), using personal smart phones/watches was used as a more realistic alternative to the generated data set to show how the proposed algorithm could be applied in a different context. [114]. We use the provided 225-length feature vectors of time and frequency domain variables generated for each instance.

- **Fashion-MNIST[6]:** This is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes (e.g., Dresses, Coat, Bag). Each class has the same number of examples. We follow a non-IID setting as in *FedAvg* [116] and divide the data into 20 parts according to their labels. We first sort the data by category label, divide each category into 4 different sizes $\{2000, 2750, 3250, 4000\}$, and assign each of 20 parts 2 different sizes. We model each part as a separate client and predict the target labels.

- **Cifar-10[7]:** This dataset contains 60,000 images associated with a label from 10 classes (e.g., airplane, dog, cat). Each class has the same number of examples. There are 50,000 training samples and 10,000 test samples. Each image is a 32x32 colour image. We follow a non-IID setting as in *FedAvg* [116] and divide the data into 20 parts according to their labels. We first sort the data by category label, divide each category into 4 different sizes $\{1500, 2250, 2750, 3500\}$, and assign each of 20 parts 2 different sizes. We model each part as a separate client and predict the target labels.

6.3.2 Comparative Methods

We compare the proposed approach with the following synchronous and asynchronous federated learning approaches.

[5] http://extrasensory.ucsd.edu/
[6] https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/
[7] https://www.cs.toronto.edu/ kriz/cifar.html

- FedAvg [5, 116]: the commonly used synchronous federated learning approach proposed by McMahan *et al.* [116].
- FedProx [134]: synchronous federated learning framework with a proximal term on the local objective function to mitigate the data heterogeneity problem and to improve the model stability compared to FedAvg.
- ASO-Fed [133]: asynchronous federated learning framework to deal with non-IID streaming device data.

6.3.3 Experimental Setup

Drift Simulation

Real-world data satisfy the gradual drift situation as these data are generated by sensors in the non-stationary environment. For instance, data on each weather sensor of the Air Quality Dataset changes from season to season. Thus we use the three provided real-world datasets to test the algorithm for gradual drift. To make the problem more challenging, following [137], we simulate the sudden concept drift in time-series datasets. First we randomly select a portion C of devices to add concept drifts. Then we modify the feature values to be random values between $10 - 1000$ of a consecutive time period of samples and keep the according labels unchanged. We perform this sudden concept drift simulation on both the FitRec dataset and Air Quality Dataset.

There are drawbacks for real-world data for evaluating concept drift handling methods, as the precise start and end time of drifts is unknown. Due to these limitations, it is difficult to evaluate methods and understand the drift [138]. Thus we also simulate concept drifts on the image data. We use the similar strategy as Mansukhani et al. [139], which is, adding noise to a portion of the image data. Same as the time-series datasets, we randomly select a portion C of participants to add noise. We set C as 0.1, which is, there are 10% devices with concept drifts in each dataset. To further evaluate the model performance with this parameter changes, we report the results with different values of C in Section 6.4.3.

Table 6.1: Prediction performance and statistics of all methods on four datasets. 'Avg' is the average prediction performance on all devices; 'Var' shows the variance of the final prediction performance distribution of devices. We show the results of Air Quality data on 20% devices with concept drift due to this dataset has a smaller number of devices.

		Cifar-10 (Accuracy↑)	Fashion-Mnist (Accuracy↑)	FitRec (Smape ↓)	Air Quality (Smape ↓)	ExtraSensory (F1-score↑)
Avg (all devices)	FedAvg	0.822	0.913	0.897	0.446	0.527
	FedProx	0.841	0.912	0.857	0.443	0.567
	ASO-Fed	0.892	0.943	0.832	0.470	0.711
	FedConD)	**0.907**	**0.953**	**0.822**	**0.430**	**0.721**
Var (all devices)	FedAvg	1.65e-05	2.31e-05	0.0768	1.75e-3	0.0475
	FedProx	2.42e-04	6.79e-04	0.0894	2.80e-3	0.0369
	ASO-Fed	1.58e-05	4.21e-06	0.0668	6.75e-04	0.0261
	FedConD	**1.09e-05**	**2.38e-06**	**0.0671**	**2.51e-04**	**0.0226**
Var (drift devices)	FedAvg	3.59e-7	1.10e-4	1.57e-1	9.05e-4	–
	FedProx	0.2e-05	3.10e-3	5.40e-4	2.92e-3	–
	ASO-Fed	0.39e-4	0.4e-05	8.40e-4	2.01e-4	–
	FedConD	**0.85e-07**	**0.4e-05**	**1.80e-4**	**0.11e-4**	–

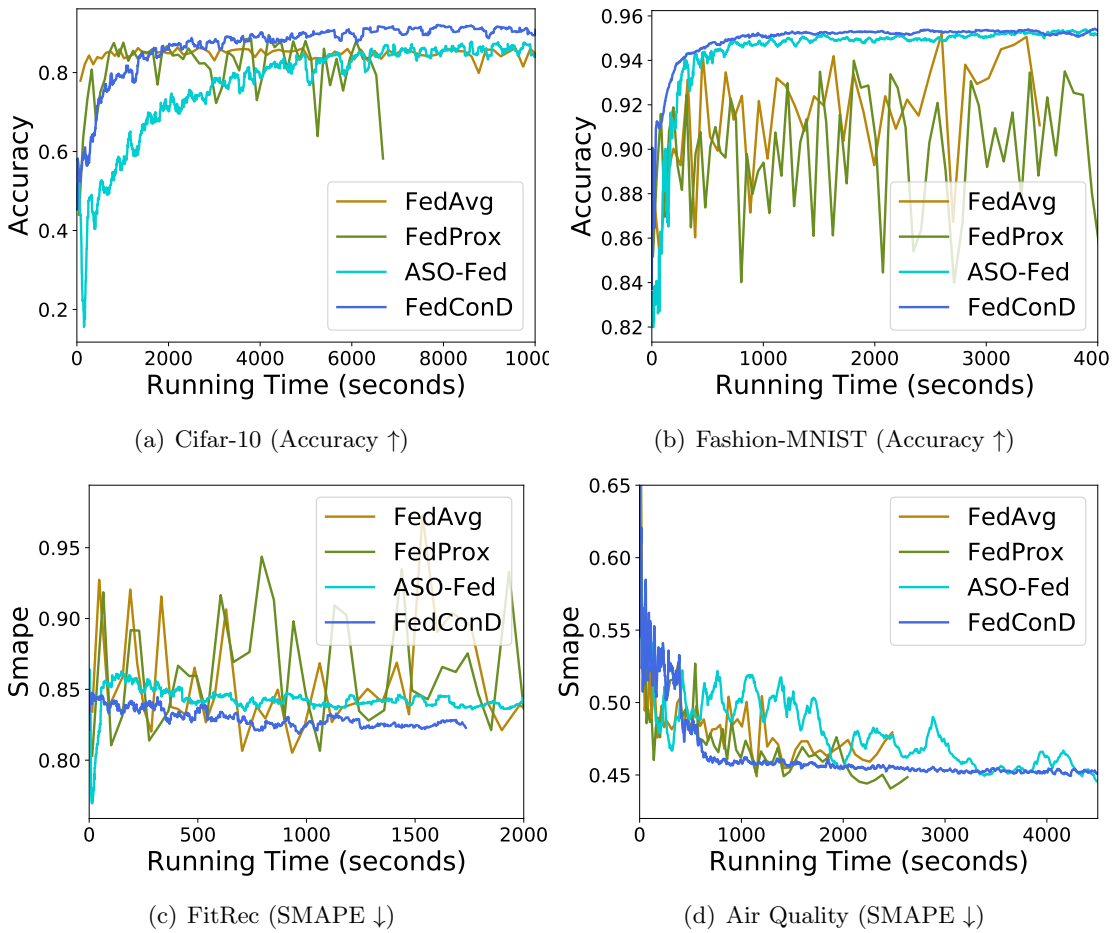
Training Details

We split the each device's data into 60%, 20%, 20% for training, validation, and testing, respectively. As for each training data, we start with a random portion of the total training size, and increase by $0.01\% - 0.1\%$ each iteration to simulate the arriving data. We set the threshold δ value is 0.2 for image classification data and 0.25 for time-series regression data. We set the portion of participation devices of FedAvg and FedProx as 0.2 at each round. We use a single layer LSTM followed by a fully connected layer for the two time-series datasets and two CNN layers followed by a max pooling layer for the two image datasets. The local epoch number of each client is set as 2. We design simple network architectures for all datesets so that it can be easily handled by mobile devices. All of the experiments are conducted with two Intel E5-2660 v3 10-core CPU at 2.60GHz [132].

6.4 Experimental Results

6.4.1 Performance Comparison

Table 6.1 reports the prediction performance comparing the proposed model with the baseline approaches. We observer that FedConD receives the best predictive performance (Avg)



(a) Cifar-10 (Accuracy ↑)

(b) Fashion-MNIST (Accuracy ↑)

(c) FitRec (SMAPE ↓)

(d) Air Quality (SMAPE ↓)

Figure 6.2: Test set performance vs. running time for four datasets. Lower SMAPE value indicates better model performance. For the synchronized federated frameworks, we plot results of *FedAvg* and *FedProx* at every 5 global iterations.

and the lowest variance (Var) among all the FL approaches. These results indicate that FedConD performs well with concept drift on local devices. As the server dynamically requests updates from local devices based on their update frequency, our framework can balance the updates among all devices and learn a more fair optimal global model. We notice that ASO-Fed has the close performance as the proposed model both on the predictive performance and the statistic results. ASO-Fed designs an online learning algorithm with a decay coefficient to balance the previous and current local gradients, which also works on

Table 6.2: Statistics of the test performance for proposed FedConDon Cifar-10 and FitRec data. FedConD can improve the test performance (Avg) on the bottom 20% devices without hurting the test performance on the top 20% devices. The variance (Var) of the final performance distribution maintains the lowest value of FedConD.

Model	Cifar-10(Accuracy↑)				FitRec(Smape↓)			
	Top 20%		Bottom 20%		Top 20%		Bottom 20%	
	Avg	Var	Avg	Var	Avg	Var	Avg	Var
FedAvg	0.823	2.62e-04	0.817	6.11e-04	0.381	1.42e-02	1.07	0.92e-03
FedProx	0.830	0.13e-04	0.821	0.12e-04	0.668	6.93e-02	0.932	1.35e-02
ASO-Fed	0.899	0.61e-05	0.868	3.51e-06	0.624	**1.10e-02**	0.879	6.57e-02
FedConD	**0.906**	**0.34e-05**	**0.889**	**7.32e-07**	**0.601**	1.21e-02	**0.831**	**0.55e-03**

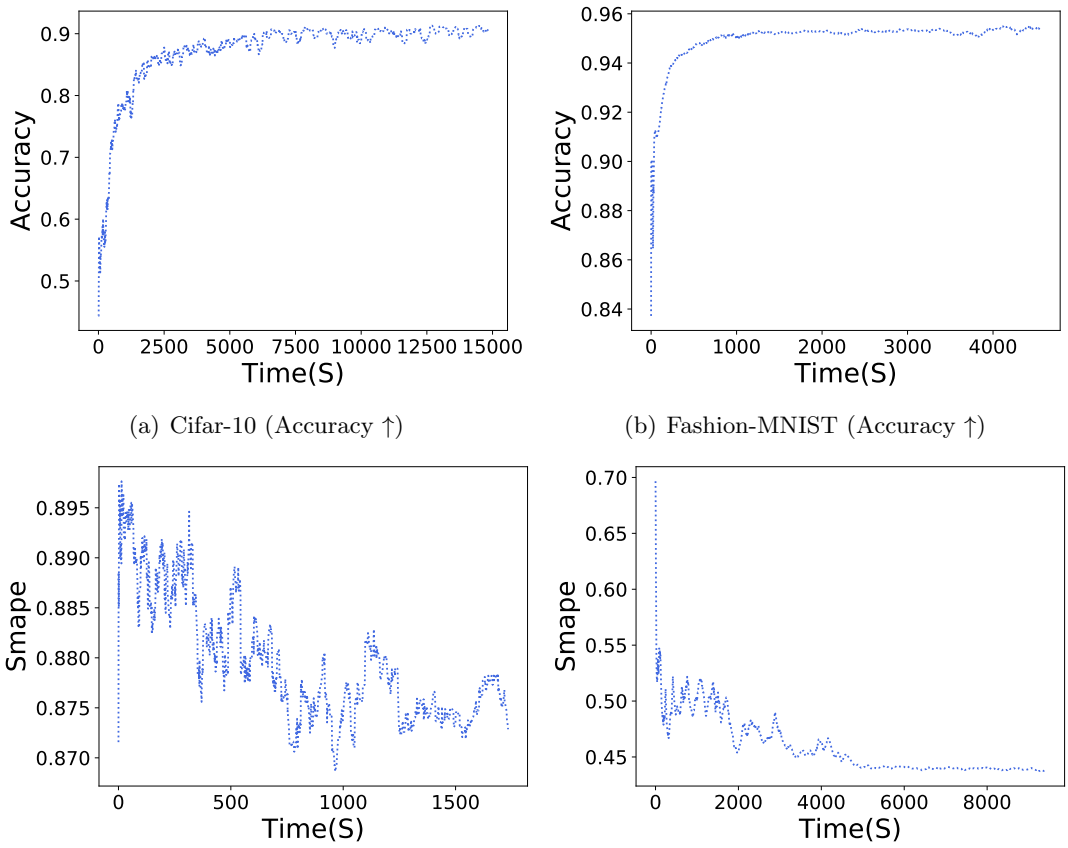
dealing with concept drifts. However, with a fixed regularization parameter across all devices, ASO-Fed is not flexible enough to handle drifts efficiently. Synchronous FLs (FedAvg and FedProx) do not perform well compared with the other two asynchronous approaches.

To evaluate the proposed drift adaption strategy, we also report the statistics on devices with drifts in Table 6.1. Each device in ExtraSensory data contains mixed drift types, thus we only report the variance across all devices. The results indicates that the proposed model can get the lowest variance within the drift devices.

Communication Efficiency

Figure 6.2 shows the learning curves of the global model on all FL frameworks. For Cifar-10 and Fashion-MNIST image data classification tasks, our proposed model achieves the best predictive performance and also converges faster than the other three approaches. For FitRec and Air Quality time-series data regression tasks, we observe larger fluctuations on ASO-Fed at the initial learning phase. Due to the high non-IID nature of these two datasets, it is common to see this fluctuations in asynchronous update frameworks as the server aggregates local updates one at each time. Our proposed framework FedConD converges more quickly and sees stable performance. The two synchronous frameworks, however, suffer from unstable model performance and cannot handle streaming data.

We further compare the two asynchronous FL methods with respect to the *communicated*



(a) Cifar-10 (Accuracy ↑)

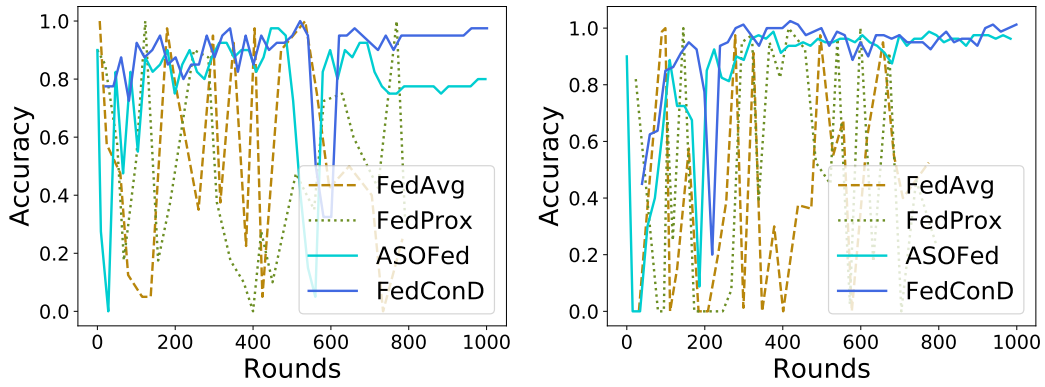
(b) Fashion-MNIST (Accuracy ↑)

(c) FitRec (SMAPE ↓)

(d) Air Quality (SMAPE ↓)

Figure 6.3: Server model evaluation performance of FedConD on four datasets. We plot results of *FedAvg* and *FedProx* at every 5 global iterations.

bytes required to achieve a certain target accuracy on a federated learning task. Table 6.3 shows the amount of upstream and downstream communications required to achieve the target prediction performance for the asynchronous FL methods. FedConDmanages to achieve the desired target performance within the smallest upload communication budget compared with ASOFed. Meanwhile, we see much smaller communication costs required for FedConD for the downstream communications. This is due to the design of γ to control the number of local training devices. We further discuss the selection of γ in Section 6.4.3.



(a) Fashion-MNIST (Accuracy ↑)

(b) Fashion-MNIST (Accuracy ↑)

Figure 6.4: Evaluation performance of local devices which have concept drift of Fashion-MNIST.

Fairness of Client Participation

The server model aims to get a equal contributions from the local clients by adjusting the update frequency of local models. While the FL developer may have different fairness criteria, achieving a more fair accuracy distribution across devices is mostly the paramount [19]. We follow the popular metrics to measure the fairness of model accuracy for different strategies [19, 140]. We evaluate the statistical performance of the test accuracy for FedConD on the best 20% and worst 20% devices. As shown in Table 6.2, FedConD can improve the predictive performance of the worst 20% devices while also maintains the best test performance for the top 20% devices. This indicates that the proposed communication-efficient strategy on the server can lead to a balanced and fair model. Lacking methods to deal with streaming data and concept drift, FedAvg and FedProx are observed to have large gaps in the average performance between the top devices and bottom devices on the studied benchmarks.

Table 6.3: *Total Bytes* required for upload(Uplink) and download (Downlink) to achieve a certain target performance on different learning tasks. Note the model size within the same learning tasks is the same across all methods.

	Model	ASO-Fed	FedConD
Cifar-10	Uplink	15124.9 MB	3708.2 MB
(Acc.: 0.75)	Downlink	302321.8 MB	14832.5 MB
Fashion-MNIST	Uplink	3173.7 MB	2386.1 MB
(Acc.: 0.94)	Downlink	63462.4 MB	9547.5 MB
Air Quality	Uplink	20.14 MB	13.88 MB
(SMAPE: 0.46)	Downlink	199 MB	27.76 MB
ExtraSensory	Uplink	698.6 MB	574.4 MB
(F1-score: 0.60)	Downlink	41910.3 MB	6893.2 MB

6.4.2 Analysis on Concept Drift

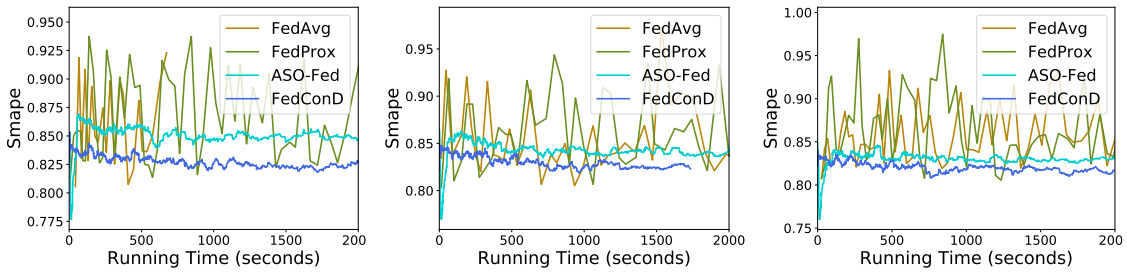
Adaptation to new concepts.

We simulate sudden drift on the image benchmark data by injecting noise on local devices (experimental protocol described in Section 6.3.3). As shown in Figure 6.2, FedConD achieves the best classification performance on Cifar-10 and Fashion-MNIST data compared with other competitors It is also noticeable that FedAvg and FedProx fail to converge in this scenario. ASO-Fed has the similar accuracy trend as FedConD, but it needs longer time to converge. This is because ASO-Fed has extra computation such as feature learning component on the server. Streaming data usually contains mixed type of concept drifts.

Adaptation to mixed concepts.

We also add sudden drift to the FitRec and Air Quality data to create the complicate drift types with a mix of gradual drift and sudden drift. We notice that the prediction performance of all FL algorithms have larger fluctuations with the mixed drifts. ASO-Fed has noticeable unstable performance at the beginning stage of the training process. FedAvg and FedProx still show no convergence trend. FedConD converges to the best performance with the lowest prediction errors.

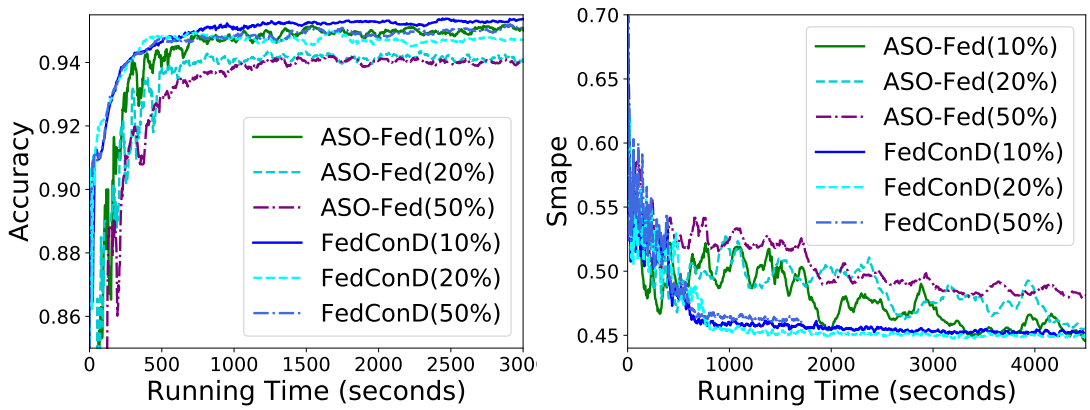
To better evaluate the effect of concept drift on both the global model and local models, we further show the predictive performance of the global model for FedConD in Figure 6.3 and local models for FedConD in Figure 6.4. As shown in Figure 6.3, FitRec is a non-IID dataset. Each local device contains data of only one sport type (e.g., hiking, biking). Thus the unstable learning curve is not only due to concept drifts but also caused by the non-IID aspects of this data. We observe the obvious drop of the prediction performance on Fashion-MNIST at the initial learning stage. This dataset is simpler than Cifar-10 for it contains only grayscale images, thus the changes on local data distributions can be reflected on the global model. In contrast, drifts on local data cannot be easily noticed on the server side for the Cifar-10 data. Air Quality data has a smaller number of devices, changing even one device's data distribution will have relatively longer affect on the global model.



(a) Early Stage Drift (b) Middle Stage Drift (c) Late Stage Drift

Figure 6.5: Convergence of FedConD compared with baseline methods for concept drift at 'Early Stage (10%)', 'Middle Stage (40%)' and 'Late Stage (70%)' on the FitRec dataset.

We show the predictive performance at each round on the drifted devices for the Fashion-MNIST dataset in Figure 6.4. Obvious drops can be noticed from the learning curve of FedConD, which indicate the data distribution changes on these local devices and the proposed model can detect these changes. ASO-Fed has similar learning trend as FedConD as it has a local decay strategy to deal with online learning, while FedConD has higher accuracy when drift occurs and converges to better performance. FedAvg and FedProx fail to detect the local drift.



(a) Fashion-MNIST (Accuracy ↑)

(b) Air Quality (SMAPE ↓)

Figure 6.6: Convergence of FedConD compared with ASO-Fed on different portion of drift devices.

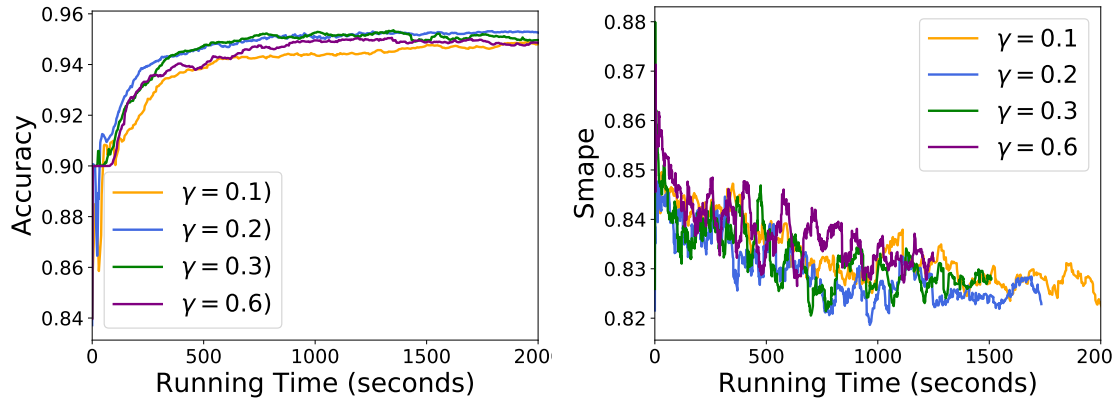
Concept Drift at Different Stages.

We perform another empirical study on model performance with drifts occurring at different stages. Figure 6.5 shows the prediction performance of FedConD compared with other FL algorithms for the sudden drift occurs at 10% place (Early Stage), 40% place (Middle Stage) and 70% place (Late Stage) of the device data. ASO-Fed has larger errors than FedConD when the drift happens at an early time, and this error gap shrinks as the drift occurs at later times. Synchronous FL algorithms have unstable prediction performance no matter when the drift starts. We also observe that the prediction errors drop as the drift starts late, which indicates a later drift affects the server model less.

6.4.3 Sensitivity to Tunable Parameters

Number of Drift Devices

We set the portion of drift devices to be 10% for the default experimental setting. In this section we also perform evaluations on the model performance with more devices which have concept drifts. As shown in Figure 6.6, we show the convergence of FedConD compared



(a) Fashion-MNIST (Accuracy ↑)

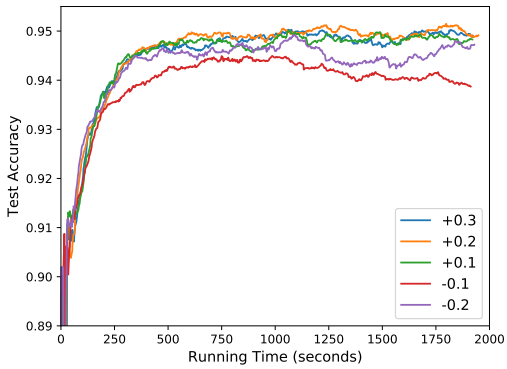
(b) FitRec (Smape ↓)

Figure 6.7: For different fraction of local training devices (γ) simultaneously, the convergence of FedConD varies. We increase *gamma* from 0.1 to 0.6, the best model performance appears at $gamma = 0.2$.

to ASO-Fed with the number of drift devices increase. With 20% drift devices, slight decrease in the prediction performance for both approaches are observed for the Fashion-MNIST data. For the Air Quality data, the SMAPE value increases a little of ASO-Fed compared with 10% drift devices setting, while there is almost no difference on the prediction performance of FedConDon 10% and 20% drift devices. We observe the similar situation for 50% drift devices, where there are slight decrease on the prediction performance and the learning curves remain stable of FedConD. These results also prove the robustness of FedConD on dealing with local concept drifts across an increasing number of devices.

Number of Local Training Devices Simultaneously

Asynchronous learning framework allows local devices to perform local training simultaneously, which is, local device can start its own training any time when it is ready. However, with all of local devices performing local training at the same time and upload the local models to the server, the server can easily become a bottleneck. Meanwhile, devices with better resources (e.g., network, hardware) will perform more updates to the server and lead to a biased global model. To deal with these problems, we design a parameter γ to control



(a) Fashion-MNIST (Accuracy ↑)

Figure 6.8: Convergence trend of FedConD on increasing or decreasing the regularization parameter λ to handle concept drift. '+' means increasing λ and '-' means decreasing λ.

the number of local active devices at the same time. Higher value of γ implies more devices are performing the local training at the same time and lower value of γ implies fewer devices perform local training simultaneously. In Figure 6.7, we show the convergence trend of FedConD with different γ values. FedConD achieves the best predictive performance when $\gamma = 0.2$ or $\gamma = 0.3$. However, with $\gamma = 0.2$, FedConD converges faster, thus we use $\gamma = 0.2$ in the experiments of FedConD.

Regularization Parameter λ

Figure 6.8 displays the convergence trend of FedConD for changing the penalty on local model with concept drifts. Intuitively, the local model will deviate further from the global model if concept drift occurs. These results confirms our claim in Section 6.2.2. When a concept drift is detected on local device, we increase the penalty to force the local model to be closer to the global model. In Figure 6.8, the model performance is better when the regularization parameter, λ is increased. Otherwise, the local model will deviate further from the global model and hurt the predictive performance on the global model.

6.5 Conclusion

In this paper, we propose a novel framework, FedConD, to detect and handle concept drift in asynchronous federated learning. To the best of our knowledge, this is the first study of concept drift for the federated learning framework with heterogeneous device data. On the server side, in order to get a more fair global model and reduce the overall communication cost, we design a strategy to balance the local updates and control the number of devices which perform local training at the same time. Experimental results on three evolving streaming data and two image data show that the proposed model can detect and handle concept drift in asynchronous federated learning efficiently.

Chapter 7: FedAT: A High-Performance and Communication-Efficient Federated Learning System with Asynchronous Tiers

This is a co-authored study and the lead author is Zheng Chai[1]. I summarized the key idea, theoretical analysis and part of the experimental results here. The complete preprint version of this study can be obtained here [35].

This study aims to overcome the deficiencies of stragglers (in synchronous FLs) and communication bottleneck (in asynchronous FLs) problems in federated learning. The proposed model, FedAT, is a novel communication-efficient FL approach that combines the best of both worlds – synchronous and asynchronous FL training – using a tiering mechanism. In FedAT, the clients involved in a FL deployment are partitioned into logical tiers based on their response latency (i.e., the time a client takes to finish a single round of training). All the logical tiers in FedAT participate in the global training simultaneously, with each tier proceeding at its own pace. Clients within a single tier update a model associated with that particular tier in a synchronous manner, while each tier, as a logical, coarse-grained, training entity, updates a global model asynchronously. Faster tiers, with shorter per-round response latency, drive the global model training to converge faster; slower tiers get involved in the global training by asynchronously sending the model updates to the server, so as to further improve the model's prediction performance. Uniformly aggregating the asynchronously updated tier model into the global model may result in biased training (biased towards the faster tiers), as more performant tiers tend to update the global model more frequently than the slower tiers. To solve this issue, I propose a new weighted aggregation heuristic, which assign higher weight to slower tiers. Furthermore, to minimize the communication

[1]https://scholar.google.com/citations?user=z3FfVt0AAAAJ

cost introduced by asynchronous training, FedAT compresses the model data transferred between the clients and the server using Encoded Polyline Algorithm. In a nutshell, FedAT synergizes the four components together, namely, the tiering scheme, asynchronous inter-tier model updating, the weighted aggregation method, and polyline encoding compression algorithm, to maximize both the convergence speed and the prediction performance while minimizing communication cost. The work presented in this chapter has been submitted to a conference and under review.

7.1 Background and Definition

In FLs, slow clients (stragglers), which perform local training at a relatively slower speed (due to weaker computing resources and/or larger local data size), may have poor prediction performance due to less training, and thus may prevent the shared model from converge to a global optimal solution. Therefore, to achieve the global optimal solution in this synchronous manner can implicitly introduce high variance in prediction performance given the existence of straggling clients. To better evaluate the robustness of FL training with stragglers, we define new metrics as a measure of the straggler tolerance level in a typical FL setup.

Definition 6. (*Robust training with straggling clients*). For two trained models w and w', we say that model w is more robust against straggling clients than model w', if (1) model w *converges faster* than model w'; (2) the test accuracy of model w for the K clients, $\{P_1, ..., P_K\}$, exhibits *lower variance* than that of model w' for the same set of K clients (where P_c represents the test accuracy of the model w over the testing data for client c), and (3) model w has better prediction performance than model w'.

As discussed in the previous sections, the existence of stragglers cause longer training times and prevent the model from converging to the optimal solution. The rational of using these three metrics, namely, convergence speed, accuracy variance, and prediction accuracy, as a means to quantify the robustness of a FL approach against stragglers, is that: (1)

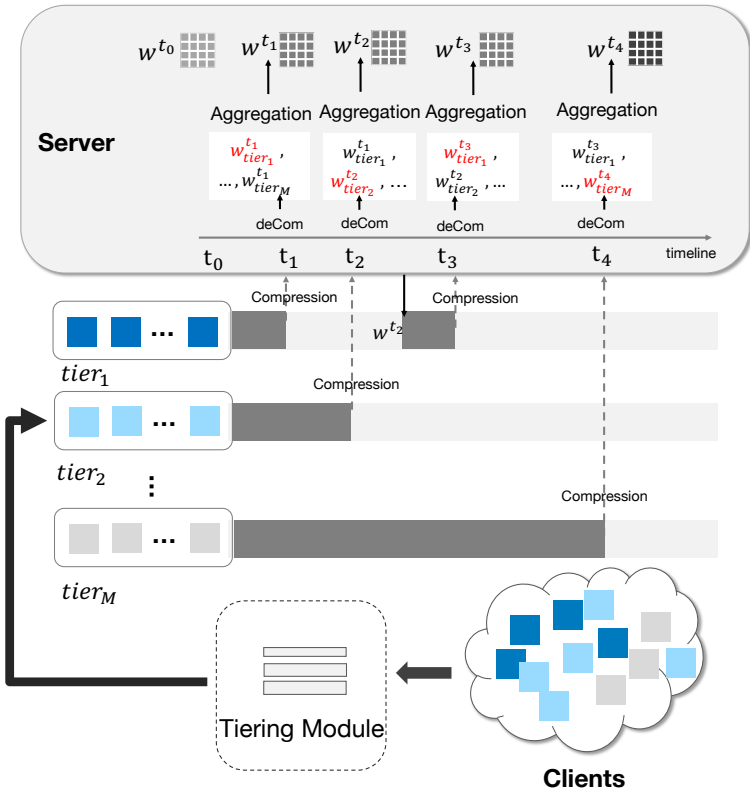


Figure 7.1: Overview of FedAT. $tier_1, ..., tier_M$ are M tiers, and $w^t_{tier_1}, ..., w^t_{tier_M}$ are their according weights, respectively. `deCom` denotes the decompression process of clients' models in a certain tier on the server.

stragglers not only cause slow convergence speed, but also a loss in prediction performance, and (2) existing literature fail to comprehensively consider all these aspects [24, 25, 34, 85, 87, 122].

7.2 FedAT: Federated Learning with Asynchronous Tiers

FedAT consists of three main components: (1) a centralized server for global model synchronization; (2) a group of clients that are logically partitioned into different performance tiers; and (3) a tiering module that profiles clients' training performance and performs client tiering based on the response latency of each client.

We next illustrate the FL training process of FedAT. The tiering module profiles and

partitions the involved clients into M tiers based on their response latencies: $\{tier_1, tier_2, ..., tier_M\}$, where $tier_1$ is the fastest tier and $tier_M$ is the slowest tier. The server maintains a list of M models, $\{w^t_{tier_1}, ..., w^t_{tier_M}\}$, one for each tier, reflecting the most updated view of per-tier local models, at a certain round t. Correspondingly, the server also maintains a global model w that gets asynchronously updated from M tiers.

Each tier performs synchronous update process, a fraction of clients ($\mathcal{S}$) are selected randomly and compute the gradient of the loss on their local data, then send the compressed weights to the server for a synchronous and update the tier model on server.

Figure 7.1 shows an example of the intra-tier synchronous and cross-tier asynchronous training process. At time t_1, the clients in $tier_1$ quickly finish their local training, compress their trained models and send to the server. The server then performs the following steps: (1) decompresses the local models received from $tier_1$, (2) applies synchronous update to the received models of $tier_1$ and get $w^{t_1}_{tier_1}$ (highlighted in red color in Figure 7.1), and (3) aggregates the latest updates sent from all the tiers (including $tier_1$) using a weighted average aggregation method, to generate a new global model w^{t_1}.

At time t_2, the last client of $tier_2$ finishes its local training and sends the compressed model to the server. The server follows the same procedure as $tier_1$ to get a new global model w^{t_2}. Then the server sends the latest global model w^{t_2} to the next ready tier (in this example $tier_1$) and starts the next round. Note that a tier in FedAT directly interacts with the server to update the global model whenever it finishes a round of local training, thus forming an *asynchronous*, cross-tier process.

Since clients are partitioned into tiers based on their response latencies and the tiers asynchronously update the global mode, stragglers may not become a performance bottleneck that would otherwise slow down the global training progress. However, as the server interacts more frequently with the faster tiers than with the relatively slower ones, this would inevitably lead to biases towards the faster tiers. To address this issue, we introduce a new objective onto the server-side optimization, which uses a weighted aggregation strategy to more fairly balance the mode updating processes from different tiers.

7.2.1 Cross-Tier Weighted Aggregation

Assuming there are M tiers, the number of updates from each tier till now is $T_{tier_1}, T_{tier_2}, ..., T_{tier_M}$, respectively, and the total number of updates from all the tiers till now is $T_{tier_1} + T_{tier_2} + ... + T_{tier_M} = T$, we define the objective function of FedAT as:

$$f(w) = \sum_{m=1}^{M} \frac{T_{tier_{(M+1-m)}}}{T} f_{tier_m}(w), \tag{7.1}$$

where $\frac{T_{tier_{(M+1-m)}}}{T}$ is the relative weight of $tier_m$, and $\sum_{m=1}^{M} \frac{T_{tier_{(M+1-m)}}}{T} = 1$. To understand the heuristic, a relatively slower tier with a tier number m would get assigned a relatively larger weight value, $\frac{T_{tier_{(M+1-m)}}}{T}$, as $M + 1 - m$ corresponds to a relatively faster tier, $tier_{(M+1-m)}$, whose historical updating frequency $T_{tier_{(M+1-m)}}$ is expected to be higher. In this way, FedAT can dynamically steer and balance the global model training, avoid potential bias towards a subset of faster tiers, and effectively improve the convergence rate.

$f_{tier_m}(w)$ is the weighted average of the models from the selected clients within $tier_m$. Assuming at round t, $tier_m$ happens to be in communication with the server, we have the update of $f_{tier_m}(w)$ as follow:

$$f_{tier_m}(w) = \sum_{k=1}^{|\mathcal{S}_t|} \frac{n_k}{N_c} F_k(w), \tag{7.2}$$

where $\mathcal{S}_t$, $|\mathcal{S}_t|$, and N_c denote a subset of randomly selected clients in $tier_m$, the number of selected clients in $tier_m$, and the total number of data samples in $\mathcal{S}_t$, respectively.

7.2.2 Training at Local Clients

As mentioned in preliminaries, for training with Non-i.i.d. data, frequent local updates may potentially cause the local models to diverge due to the varying updating frequency of different tiers and the underlying data heterogeneity. We add a constraint term to the local subproblem by restricting the local updates to be closer to the global model. Thus, instead of just minimizing the local function F_k , client k applies an update with the constraint using the following surrogate objective h_k:

$$h_k(w_k) = F_k(w_k) + \frac{\lambda}{2}||w_k - w||^2, \tag{7.3}$$

where w_k, w are the local model of client k and server model, respectively. Then we will update (7.2) as:

$$\begin{aligned} f_{tier_m}(w) &= \sum_{k=1}^{|\mathcal{S}_t|} \frac{n_k}{N_c} h_k(w_k) \\ &= \sum_{k=1}^{|\mathcal{S}_t|} \frac{n_k}{N_c} (F_k(w_k) + \frac{\lambda}{2}||w_k - w||^2). \end{aligned} \tag{7.4}$$

7.3 Convergence Analysis

In this section, we show that FedConD converges to the global optimal solution on Non-i.i.d. data.

7.3.1 Convergence for Convex Functions

First, we introduce some definitions and assumptions as follow.

Definition 7. (Smoothness) The function f has Lipschitz continuous gradients with constant $L > 0$ (in other words, f is *L-smooth*) if $\forall x_1, x_2$,

$$f(x_1) - f(x_2) \leq \langle \nabla f(x_2), x_1 - x_2 \rangle + \frac{L}{2}||x_1 - x_2||^2. \tag{7.5}$$

Definition 8. (Strong convexity) The function f is *μ-strongly convex* with $\mu > 0$ if $\forall x_1, x_2$,

$$f(x_1) - f(x_2) \geq \langle \nabla f(x_2), x_1 - x_2 \rangle + \frac{\mu}{2}||x_1 - x_2||^2. \tag{7.6}$$

Definition 9 has been made by the work [134].

Definition 9. (γ-inexactness) For a function $h(w) = F(w) + \frac{\lambda}{2}||w - w_0||^2$, and $\gamma \in [0, 1]$. w^* is a γ-inexact solution for $\min_w h(w)$ if $||\nabla h(w^*)|| \leq \gamma ||\nabla h(w_0)||$, where $\nabla h(w) = \nabla F(w) + \lambda(w - w_0)$.

According to [141], this definition aims to allow flexible performance of local clients in each communication round, such that each of the local objectives can be solved inexactly. The amount of local computation vs. communication can be tuned by adjusting the number of local iterations, i.e., more local iterations indicates more exact local solutions.

Further, we make the following assumptions on the objective functions:

Assumption 2. *The central objective $f(w)$ is bounded, i.e.,* $\min f(w) = f(w_*) > -\infty$

Assumption 3. *The expected squared norm of stochastic gradients is uniformly bounded, i.e., there exists a scalar G, such that $\mathbb{E}||\nabla F(w^t)||^2 \leq G^2$, all $t = 0, ..., T-1$.*

Assumption 4. *With $\bar{g}_t(w^t)$ ($\bar{g}_t = \sum_{k=1}^{m} \frac{n_k}{N_c} \nabla h_k(w^t)$) as the averaged gradients from certain tier with m clients, there exists a scalar $\sigma > 0$ such that $\nabla f(w^t)^\top \mathbb{E}(\bar{g}_t(w^t)) \geq \sigma ||\nabla f(w^t)||^2$.*

Assumption 2 is easy to satisfy as there exists a minimum value for the central objective $f(w)$. The conditions in Assumption 3 on the variance of stochastic gradients is customary.

While this is a much weaker assumption compared to the one that uniformly bounds the expected norm of the stochastic gradient. Assumption 4 ensure that the gradient of local tier $\bar{g}_t$ is an estimation of $\nabla f(w^t)$. And as $\sigma = 1$, we have $\bar{g}_t$ as the unbiased estimation of $\nabla f(w^t)$.

To convey our proof clearly, we first introduce and prove certain useful lemmas.

Lemma 2. *With Definition 9, the local functions $h(\cdot)$ are γ-inexact. For aggregated tier model $\bar{g}_t(w^t)$, we have*

$$\mathbb{E}||\bar{g}_t(w^t)||^2 \leq \gamma^2 G^2 c^2, \tag{7.7}$$

where c is the total number of clients within the given tier.

Lemma 3. *If $f(w)$ is μ-strongly convex, then with Definition 8, we have:*

$$2\mu(f(w^t) - f(w_*)) \leq ||\nabla f(w^t)||^2. \tag{7.8}$$

We show that the averaged model of local tier is bounded in Lemma 2. The detailed proof is in Appendix 7.6.1. While the proof of Lemma 3 is supported by the literature [130, 131], we also provide a detailed proof in Appendix 7.6.2.

Theorem 3. *Suppose that the central objective function $f(w)$ is L-smooth and μ-strongly convex. The local functions $h(\cdot)$ are γ-inexact. Let Assumption 3 and Assumption 4 hold. After T global updates on the server, FedAT converges to a global optimum w_*:*

$$\begin{aligned} &\mathbb{E}[f(w^T) - f(w_*)] \\ &= (1 - 2\mu B\eta\sigma)^T (f(w^0) - f(w_*)) + \frac{L}{2}\eta^2\gamma^2 B^2 G^2 c^2, \end{aligned} \tag{7.9}$$

where c is the total number of clients within one tier, and $B = \frac{T_{tier_{(M+1-m)}}}{T} \leq 1$.

We direct the reader to Appendix 7.6.3 for a detailed proof. The convergence bound in Theorem 3 depends on local constrain μ, weighted parameter B and learning rate η. Note

that B varies in each global iteration because the update number of each tier changes at every global iteration. A also varies due to the data heterogeneity of each tier.

7.3.2 Convergence for Other Situations

Theorem 4. *Suppose that the central objective function $f(w)$ is L-smooth and non-convex. The local functions $h(\cdot)$ are γ-inexact. Let Assumption 2,Assumption 3 and Assumption 4 hold, then after T global updates we have:*

$$\begin{aligned} &\sum_{t=0}^{T-1} B\mathbb{E}[||\nabla f(w_t)||^2] \\ &\leq \frac{f(w^0) - f(w_*)}{\eta\sigma} + \frac{L}{2\sigma} T^2 \eta\gamma^2 B G^2 c^2, \end{aligned} \tag{7.10}$$

where c is the total number of clients within one tier, and $B = \frac{T_{tier_{(M+1-m)}}}{T} \leq 1$.

7.4 Evaluation

7.4.1 Experimental Setting

Federated Datasets. We evaluate FedAT using three different real-world federated datasets on both the convex and non-convex models described as follows:

- CIFAR-10: We train a convolutional neural network (CNN) on the 10-class CIFAR-10 [142] dataset. The network architecture includes three convolutional layers, each with 32, 64 and 64 filters, followed by two fully connected layers with units of 64 and 10. We partition the dataset into 100 clients and follow the same Non-i.i.d. setting of CIFAR-10 as [116].
- Fashion-MNIST: We use a 10-class Fashion-MNIST dataset [143] to train an image classification model with the same mode structure, number of clients and Non-i.i.d.

Table 7.1: Comparison of prediction performance and variance to baseline approaches. `#class` indicates the number of labels (i.e., classes) each client has. The `Accuracy` columns show the best prediction accuracy that each FL approach reaches after each model converges. The `Norm.Var.` columns show the average variance of test accuracy among all clients, normalized to that of FedAT. We show FedAT's absolute variance values (`Abs.Var.`). We show the absolute values for FedAT's accuracy variance. `impr.(a)` and `impr.(b)` are the accuracy improvement of FedAT compared with the best and worst baseline FL method, respectively. The best performance results are highlighted in bold font.

Dataset(#class)		CIFAR-10					Fashion-MNIST	Sentiment140
		#2	#4	#6	#8	i.i.d.	#2	
TiFL	Accuracy	0.527	0.615	0.654	0.655	0.685	0.859	0.739
	Norm. Var.	1.26	2.79	1.33	1.3	2.12	1.29	2.75
FedAvg	Accuracy	0.547	0.628	0.654	0.667	0.686	0.842	0.741
	Norm. Var.	2	5.07	4.33	3.1	4.23	1.86	3.72
FedProx	Accuracy	0.509	0.609	0.624	0.650	0.669	0.831	0.742
	Norm. Var.	1.261	6.75	3.981	2.22	2.992	2.243	3.89
FedAsync	Accuracy	0.480	0.541	0.531	0.561	0.567	0.795	0.740
	Norm. Var.	2	3.93	2.08	1.54	2.69	2	5.69
FedAT	Accuracy	**0.591**	**0.633**	**0.673**	**0.681**	**0.701**	**0.873**	**0.748**
	Abs. Var.	**0.0042**	**0.0014**	**0.0012**	**0.001**	**0.00052**	**0.007**	$\mathbf{2.67e^{-5}}$
	impr.(a)	7.44%	0.79%	2.82%	2.05%	2.13%	1.6%	0.93%
	impr.(b)	18.78%	14.53%	21.09%	17.62%	19.11%	8.93%	1.2%

setting as CIFAR-10. The input of the model is a flattened 784-dimensional (28×28) image, and the output is a class label between 0 and 9, with each label corresponding to one cloth type.

- Sentiment140: In order to evaluate the model performance under a convex setting, we train a logistic regression model on a text sentiment analysis task on tweets from the Sentiment140 [144] dataset. Each twitter account corresponds to a client.

FL Methods. We compare FedAT against four synchronous and asynchronous FL methods:

- FedAvg [116]: A baseline synchronous FL method proposed by McMahan et al. At each round, a certain ratio of total clients are randomly selected for training, the server aggregates the weights received from selected clients in an average manner.

- TiFL [122]: A synchronous FL method that partitions training clients into different

tiers based on their responding latency. For each round, one tier is selected according to a novel adaptive selection strategy which is related to the test accuracies of all tiers, then certain number of clients in that tier are selected for training. The aggregation method of TiFL is adopted from FedAvg.

- FedProx [134]: A synchronous FL method that claiming to handling stragglers due to system heterogeneity across all clients by applying different local epochs for clients. Also FedProx adds proximal term to the objective on the clients for improving performance and smoothing the training curve.

- FedAsync [24]: A baseline asynchronous FL method using weighted averaging to update the server model. Different from previous synchronous FL methods, all clients train concurrently, when the server receives weights from any client, the weights are weighted averaged with current global weights to get the latest global weights, then communicate to all available clients at that time for training.

Implementation and Setup. We have implemented FedAT and the comparison methods all in TensorFlow [145]. We simulate a FL setup using a 192-core Chameleon Cloud cluster, which consists of three bare-metal machines, each equipped with a 64-core Intel® Xeon® Gold 6242 CPU, and 192 GB main memory. We deploy the FL server exclusively on one machine, and all clients on the rest two, where each client gets assigned one CPU core. We evaluate 100 clients in most tests, unless otherwise specified. FedAT is configured to use `Precision 4` as the precision of the compressor (subsubsec:compression) throughout the evaluation.

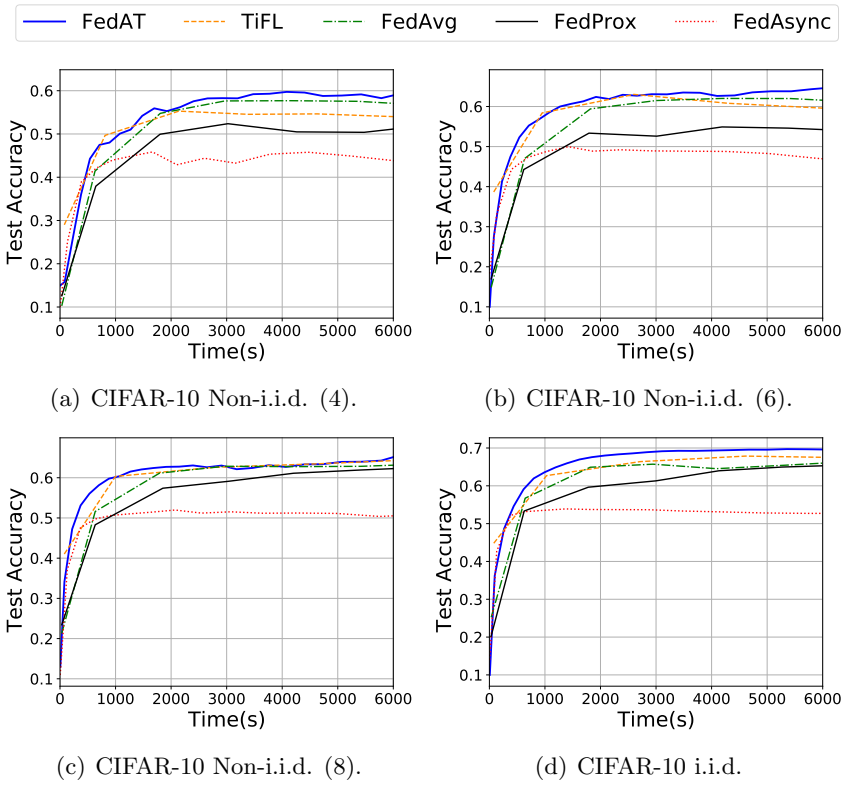
Hyperparameters. We randomly split each client's local data into an 80% training set and a 20% testing set. For intra-tier synchronous training, we adopt the same sampling scheme as FedAvg: sampling clients (within a particular tier) randomly at each round. We use Adam [117] as the local solver and set the local constrain parameter λ as 0.4. For each dataset, we tune the learning rate on FedAvg (*local epoch* $E = 3$, *batch size* $= 10$) and use

the same learning rate for all the four FL methods on that dataset. We set the number of randomly selected clients as 10 for FedAvg, TiFL, FedProx and FedAT on all datasets.

Simulating Different Performance Tiers. Clients in FL are typically edge devices, and their computing power and network connection may not be stable; hence simply assigning a fixed amount of resources is not sufficient to reflect the real situation. Therefore, we assign 1 CPU for each client during the whole training process and add random delays to the computations conducted by clients; the added random delay is to simulate different levels of straggler effects that are caused by weaker computing powers and intermittent network connections in a real-world FL setup. We first evenly divide all the clients into 5 parts, then randomly assign delays of $0s$, $0 \sim 5s$, $6 \sim 10s$, $11 \sim 15s$ and $20 \sim 30s$ to the clients in each part at every round, respectively. Each part is called one *tier*. To guarantee fair comparison, each client, once selected, would follow a fixed, pseudo-random mini-batch schedule. The same strategy is applied to all the FL methods that we test (including FedAT's intra-tier synchronous training). Furthermore, to simulate unstable network connections, for all the tests that we run, we randomly select 10 "unstable" clients, which would drop out at any time during the training process. Once the client drops out, it will not come back and rejoin the training process again.

7.4.2 Prediction Performance

Table 7.1 presents the results of the prediction performance and the variance of the test accuracy on all the datasets. We report the best test accuracy after each training process converges within a global iteration budget. For the 2-class CIFAR-10 dataset, FedAT outperforms the best baseline FL method, FedAvg, by 7.44%, and the worst baseline method, FedAsync, by 18.78%. Using the same tiering scheme as TiFL, FedAT achieves consistently higher accuracy than TiFL for all the experiments. This is because: (1) the local constraint forces local models to be closer to the server model, and (2) FedAT's new weighted aggregation heuristic can more effectively engage the straggling clients from the slower tiers,



(a) CIFAR-10 Non-i.i.d. (4).

(b) CIFAR-10 Non-i.i.d. (6).

(c) CIFAR-10 Non-i.i.d. (8).

(d) CIFAR-10 i.i.d.

Figure 7.2: Convergence speed comparison on CIFAR-10 over different level of Non-i.i.d.-ness. The results are average-smoothed for every 40 global rounds.

leading to better prediction performance (we evaluate the effectiveness of our weighted aggregation method in com-aggregation). FedAvg has the closest prediction performance as TiFL, because they both follow the same synchronous updating strategy. FedAsync, on the other hand, performs the worst, as it simply aggregates weights from one client at a round and has no effective way to deal with stragglers.

Impact of Non-i.i.d. Level.

The models' convergence behaviors are sensitive to the degree of Non-i.i.d. of the data distribution across clients. Table 7.1 shows that, for the CIFAR-10 dataset, the test accuracy increases as the degree of Non-i.i.d. decreases (i.e., the number of classes per client increases); accordingly, the variance of the test accuracy decreases as the degree of Non-i.i.d.

decreases (i.e., the data is more evenly shuffled and each client covers all the classes).

Robustness to Stragglers.

As defined in in Definition 6, the robustness of a FL method against stragglers can be quantified using the variance on the prediction performance and the convergence speed. Table 7.1 shows that FedAT has consistently the lowest accuracy variance across all experiments. FedAvg observes significantly higher accuracy variance, which are 1.86-5.07$\times$ higher than that of FedAT. This is due to the compound effect of both synchronous training and stragglers – synchronous training determines that during each round only a subset of clients can get involved to contribute to the global training, while the straggling clients are more likely to have a less accurate model when they next get selected (since they receive less training) by the server for training, thus causing huge accuracy fluctuation of the global model.

7.5 Summary

This chapter presented FedAT, a new FL method that maximizes the prediction performance and minimizes the communication cost using a tiered, hybrid synchronous-asynchronous training mechanism.

My contributions for this study include proposing to use the asynchronous update manner among tiers; and providing rigorous theoretical analysis for the proposed method for two general classes of convex and non-convex losses. I show that FedAT has provable model performance guarantee. Empirical evaluation has validated the theoretical analysis, and demonstrates that FedAT achieves the highest prediction performance compared to state-of-the-art FL methods.

7.6 Theoretical Analysis of FedAT

We analyze FedAT in the setting of both convex and non-convex situations in this section.

Recall that w^t is the model parameters of the server maintained at the t-th round. Let $\bar{g}_t(w^t) = \sum_{k=1}^{c} \frac{n_k}{N_c} \nabla h_k(w^t)$, in which, N_c is the total number of data samples across all c clients at tier m. Therefore, $w^{t+1} = w^t - \frac{T_{tier_{(M+1-m)}}}{T} \eta \bar{g}_t(w^t)$.

7.6.1 Proof of Lemma 2

To prove Theorem 3, we first introduce two Lemmas.

Proof. Using the notion of γ-inexactness for each local objective. We have

$$\nabla h_k(w^t) = F_k(w^t) + \lambda(w^t - w_0) \tag{7.11}$$

$$||\nabla h_k(w^t)|| \leq \gamma ||\nabla F_k(w^t)|| \tag{7.12}$$

With $\bar{g}_t(w^t) = \sum_{k=1}^{c} \frac{n_k}{N_c} \nabla h_k(w^t)$, we can get

$$\begin{aligned} ||\bar{g}_t(w^t)||^2 &= \frac{1}{N_c^2} ||n_1 \nabla h_1(w^t) + n_2 \nabla h_2(w^t) +, ..., + n_c \nabla h_c(w^t)||^2 \\ &\leq \frac{1}{N_c^2} \cdot N_c^2 ||\nabla h_1(w^t) + \nabla h_2(w^t) +, ..., + \nabla h_c(w^t)||^2 \\ &\leq m^2 ||\nabla h_{k^*}(w^t)||^2 \quad (k^* = \arg\max_k \nabla h_k(w^t)) \\ &\leq m^2 \gamma^2 ||\nabla F_{k^*}(w^t)||^2 \quad (with\ Eq.(7.12)) \end{aligned} \tag{7.13}$$

Take expectation of both sides and with Assumption 3, we have

$$\begin{aligned} \mathbb{E}||\bar{g}_t(w^t)||^2 &\leq m^2 \gamma^2 \mathbb{E}||\nabla F_{k^*}(w^t)||^2 \\ &\leq \gamma^2 G^2 c^2 \end{aligned} \tag{7.14}$$

□

7.6.2 Proof of Lemma 3

Proof. $f(w)$ is *μ-strongly convex*, we can get:

$$f(w') - f(w^t) \geq \langle \nabla f(w^t), w' - w^t \rangle + \frac{\mu}{2}||w' - w^t||^2, \tag{7.15}$$

Let us define $\Gamma(w')$ such that:

$$\Gamma(w') = f(w^t) + \langle \nabla f(w^t), w' - w^t \rangle + \frac{\mu}{2}||w' - w^t||^2, \tag{7.16}$$

$\Gamma(w')$ is a quadratic function of w', then it has minimal value when $\nabla\Gamma(w') = \nabla f(w^t) + \mu(w' - w^t) = 0$. Then the minimal value of $\Gamma(w')$ is obtained when $w' = w^t - \frac{\nabla f(w^t)}{\mu}$, which is:

$$\Gamma_{\min} = f(w^t) - \frac{||\nabla f(w^t)||^2}{2\mu}, \tag{7.17}$$

For $f(w)$ is *μ-strongly convex*, we can complete the proof:

$$f(w_*) \geq \Gamma(w_*) \geq \Gamma_{\min} = f(w^t) - \frac{||\nabla f(w^t)||^2}{2\mu}, \tag{7.18}$$

$$2\mu(f(w^t) - f(w_*)) \leq ||\nabla f(w^t)||^2. \tag{7.19}$$

□

7.6.3 Proof of Theorem 3

Proof. Now we start to prove the convergence of Theorem 3. With Definition 7 we can get:

$$
\begin{aligned}
& f(w^{t+1}) - f(w^t) \\
& \leq \langle \nabla f(w^t), w^{t+1} - w^t \rangle + \frac{L}{2}||w^{t+1} - w^t||^2 \qquad (f(\cdot)\ is\ L\text{-}smooth) \\
& = -\nabla f(w^t)^\top \frac{T_{tier_{(M+1-m)}}}{T} \eta \bar{g}_t(w^t) + \frac{L\eta^2}{2} \frac{T^2_{tier_{(M+1-m)}}}{T^2} ||\bar{g}_t(w^t)||^2 \qquad (w^{t+1} = w^t - \frac{T_{tier_{(M+1-m)}}}{T} \eta \bar{g}_t(w^t))
\end{aligned} \tag{7.20}
$$

Let $B = \frac{T_{tier_{(M+1-m)}}}{T}$. Then with Lemma 2, we can update Equation 7.20 as

$$
\begin{aligned}
& \mathbb{E}[f(w^{t+1})] - f(w^t) \\
& \leq -\nabla f(w^t)^\top B\eta \mathbb{E}[\bar{g}_t(w^t)] + \frac{L}{2}\eta^2 B^2 \mathbb{E}||\bar{g}_t(w^t)||^2 \\
& \leq -\nabla f(w^t)^\top B\eta \mathbb{E}[\bar{g}_t(w^t)] + \frac{L}{2}\eta^2 \gamma^2 B^2 G^2 c^2
\end{aligned} \tag{7.21}
$$

Then from Assumption 4, we have

$$
\begin{aligned}
& \mathbb{E}[f(w^{t+1})] - f(w^t) \\
& \leq -B\eta\sigma ||\nabla f(w^t)||^2 + \frac{L}{2}\eta^2 \gamma^2 B^2 G^2 c^2
\end{aligned} \tag{7.22}
$$

Then with Lemma 3, Equation (7.22) can be updated as

$$
\begin{aligned}
& \mathbb{E}[f(w^{t+1})] - f(w^t) \\
& \leq -2\mu B\eta\sigma (f(w^t) - f(w_*)) + \frac{L}{2}\eta^2 \gamma^2 B^2 G^2 c^2
\end{aligned} \tag{7.23}
$$

By subtracting $f(w_*)$ from both sides and moving $f(w^t)$ from left to right, we get

$$\begin{aligned}
&\mathbb{E}[f(w^{t+1})] - f(w_*) \\
&\leq -2\mu B\eta\sigma(f(w^t) - f(w_*)) + (f(w_t) - f(w_*)) + \frac{L}{2}\eta^2\gamma^2B^2G^2c^2 \\
&= (1 - 2\mu B\eta\sigma)(f(w^t) - f(w_*)) + \frac{L}{2}\eta^2\gamma^2B^2G^2c^2
\end{aligned} \tag{7.24}$$

Taking the whole expectations and rearranging (7.24), we obtain

$$\begin{aligned}
&\mathbb{E}[f(w^{t+1}) - f(w_*)] \\
&\leq (1 - 2\mu B\eta\sigma)\mathbb{E}[(f(w^t) - f(w_*))] + \frac{L}{2}\eta^2\gamma^2B^2G^2c^2
\end{aligned} \tag{7.25}$$

subtracting $\frac{L\eta\gamma^2BG^2m^2}{4\mu\sigma}$ from both sides, we have

$$\begin{aligned}
&\mathbb{E}[f(w^{t+1}) - f(w_*)] - \frac{L\eta\gamma^2BG^2c^2}{4\mu\sigma} \\
&\leq (1 - 2\mu B\eta\sigma)(\mathbb{E}[(f(w^t) - f(w_*))] - \frac{L\eta\gamma^2BG^2c^2}{4\mu\sigma})
\end{aligned} \tag{7.26}$$

The left side of (7.26) is a geometric series with common ratio $1 - 2\mu B\eta\sigma$, when $t + 1 = T$, we get Equation (7.9), then we complete the proof. □

7.6.4 Proof of Theorem 4

Proof. Take expectation at both sides of Equation (7.22), we have

$$\begin{aligned} &\mathbb{E}[f(w^{t+1})] - \mathbb{E}[f(w^t)] \\ &\leq -B\eta\sigma\mathbb{E}[||\nabla f(w_t)||^2] + \frac{L}{2}\eta^2\gamma^2B^2G^2c^2 \end{aligned} \tag{7.27}$$

Then sum Equation (7.27) at both sides over global iteration T. We have

$$\begin{aligned} &\mathbb{E}[f(w^{t+1})] - f(w^0) \\ &\leq \sum_{t=0}^{T-1} -B\eta\sigma\mathbb{E}[||\nabla f(w_t)||^2] + \frac{L}{2}T^2\eta^2\gamma^2B^2G^2c^2 \end{aligned} \tag{7.28}$$

As $\min f(w^t) = f(w_*) \leq \mathbb{E}[f(w^{t+1})]$, then we have

$$\begin{aligned} f(w_*) \leq f(w^0) - \sum_{t=0}^{T-1} B\eta\sigma\mathbb{E}[||\nabla f(w_t)||^2] \\ + \frac{L}{2}T^2\eta^2\gamma^2B^2G^2c^2 \end{aligned} \tag{7.29}$$

Rearrange (7.29) we can get

$$\begin{aligned} &\sum_{t=0}^{T-1} B\mathbb{E}[||\nabla f(w_t)||^2] \\ &\leq \frac{f(w^0) - f(w_*)}{B\eta\sigma} + \frac{L}{2\sigma}T^2\eta\gamma^2BG^2c^2 \end{aligned} \tag{7.30}$$

Then we complete the proof. □

Chapter 8: Conclusion and Future Work

To conclude, this thesis provide several solutions for sensor data learning with multi-task or federated learning frameworks. First of all, to learn with heterogeneous multivariate temporal data obtained from individual sensors, I propose a novel attention network M-Att for multi-task learning with sensor data. In this work I also developed two other shared deep neural networks (M-LSTM and M- CNN/LSTM) for comparison. I applied the proposed multi-task models on classification and regression tasks. The experiments show that the proposed M-Att with attention mechanism extracts shared feature representations across tasks with high flexibility. Especially in the situation when a certain task lacks training data, the attention-based multi-task model can achieve much better performance. The proposed M-Att model is robust for the multi-label and highly unbalanced class label benchmarks. Furthermore, to improve the predictive performance and learn separate models for each local devices (tasks), I present FATHOM, a federated multi-task model which can jointly trains classification/regression models from multiple distributed devices. This model utilize a hierarchical attention mechanism to generate more efficient task-specific feature representations. The task-specific attention component is designed to capture feature correlations within each local task and global temporal attention is used to generalize inter-task feature representations across all tasks. I evaluate the proposed model on both classification and regression tasks. The results show that this approach achieves improved performance compared to a wide range of state-of-the-art methods. To show the extracted key features by the attention components, I also show multiple qualitative case studies to interpret the model.

In real-life situation, multi-task learning models are not suitable for edge device training; since they assume that all clients (devices) participate in each training round. This requires that all clients are available because each client is training an individual specific

model [21]. However, edge devices could be frequently offline during the training process due to unreliable networks or other factors. Therefore, I propose a novel asynchronous online federated learning approach to tackle the learning problems on distributed edge devices. Compared to synchronized FL approaches (FedAvg [116] and FedProx [141]), ASO-Fed is computationally efficient since the central server does not need to wait for lagging clients to perform aggregation. Compared with asynchronous approach (FedAsync [24]), the proposed approach achieves better performance on all provided datasets, which indicates that the proposed asynchronous update method can better handle local streaming data. Time efficiency is compared on multiple benchmarks and the results show that the proposed ASO-Fed is faster than synchronized FLs. I also perform feature extraction on the central server and regularization at the clients to learn effective client relationships. Experimental results show that ASO-Fed can achieve close or even better performance compared to the Global baseline method. The proposed ASO-Fed inherits the idea of using asynchronous update scheme as [24, 25]. Likewise, it shares the same communication bottleneck problem pointed out in [19]. Nevertheless, it is still an open issue to build communication efficient methods in asynchronous federated learning frameworks. Besides, the underlying distribution of local device data may also change during the learning process due to these devices are usually in the non-stationary environment. To deal with these problems, I further propose another framework, FedConD, to detect and handle concept drift in asynchronous federated learning. To the best of my knowledge, this is the first study of concept drift on federated learning with heterogeneous device data. On the server side, in order to get a more fair global model and reduce the overall communication cost, I design a strategy to balance the local updates and control the number of devices which perform local training at the same time. Experimental results on four bechmark federated datasets show that the proposed model can detect and handle concept drift in asynchronous federated learning efficiently.

8.1 Future Work

8.1.1 Extending to other Machine Learning Paradigms

Most of current sensor data learning approaches are within the supervised machine learning paradigms where labels are naturally available on each device. Currently, one of the most popular frameworks used for device sensor data learning is federated learning. Extending FL to other ML paradigms, including reinforcement learning, semi- supervised learning, unsupervised learning and active learning are interesting and open challenges. This thesis provided possible solutions for FL combined with online learning (ASO-Fed and FedConD). There are more work to do to extend the FL to more applications on vary machine learning problems.

8.1.2 More Flexible Learning Procedure

Devices are not stable during the learning process due to system or network constraints. In real-life situation, devices may leave or join the training at anytime. Current FL studies on lagging or dropout devices are usually under certain assumptions (e.g., only consider the change on one condition). Combining with other issues like data heterogeneity, making the learning process more complicated. Additionally, devices which have less amount of data may need more local training. Therefore, more flexible models which allow devices to adjust their learning procedures are needed for the sensor data study.

8.1.3 Consistency on Evaluation and Benchmark

Non-IID is the one important characteristic of device data in federated learning, however, currently there is also no consensus on the metric of non-IID-ness. With more work on federated learning systems, we need a benchmark with representative data sets and workloads. Although there have been quite a few benchmarks [82], no benchmark has been widely used in the experiments for sensor data of federated learning. Therefore a robust and reliable benchmark dataset is needed for device sensor learning.

Bibliography

Bibliography

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] L. Horwitz, "The future of iot miniguide: The burgeoning iot market continues," *https://www.cisco.com/c/en/us/solutions/internet-of-things/future-of-iot.html*, 2019.

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

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

[5] ——, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[6] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[7] K. Lin, J. Xu, I. M. Baytas, S. Ji, and J. Zhou, "Multi-task feature interaction learning," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1735–1744.

[8] J. Chen, X. Qiu, P. Liu, and X. Huang, "Meta multi-task learning for sequence modeling," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] L. Han, Y. Zhang, G. Song, and K. Xie, "Encoding tree sparsity in multi-task learning: A probabilistic framework," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[10] H. Bilen and A. Vedaldi, "Integrated perception with recurrent multi-task neural networks," in *Advances in neural information processing systems*, 2016, pp. 235–243.

[11] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

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

[13] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[14] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[15] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 273–278.

[16] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.

[17] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 2016, pp. 207–212.

[18] 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.

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

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

[21] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

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

[23] T. Chen, G. Giannakis, T. Sun, and W. Yin, "Lag: Lazily aggregated gradient for communication-efficient distributed learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 5050–5060.

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

[25] M. Chen, B. Mao, and T. Ma, "Efficient and robust asynchronous federated learning with stragglers," 2019.

[26] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 59–68.

[27] S. Mehta *et al.*, "Concept drift in streaming data classification: algorithms, platforms and issues," *Procedia computer science*, vol. 122, pp. 804–811, 2017.

[28] A. Liu, Y. Song, G. Zhang, and J. Lu, "Regional concept drift detection and density synchronized drift adaptation," in *IJCAI International Joint Conference on Artificial Intelligence*, 2017.

[29] M. Sugiyama and M. Kawanabe, *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[30] A. Storkey, "When training and test sets are different: characterizing learning transfer," *Dataset shift in machine learning*, vol. 30, pp. 3–28, 2009.

[31] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[32] Y. Chen and H. Rangwala, "Attention-based multi-task learning for sensor analytics," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 2187–2196.

[33] Y. Chen, Y. Ning, Z. Chai, and H. Rangwala, "Federated multi-task learning with hierarchical attention for sensor data analytics," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

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

[35] Z. Chai, Y. Chen, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data," *arXiv preprint arXiv:2010.05958*, 2020.

[36] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2019.

[37] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," *arXiv preprint arXiv:1412.6651*, 2014.

[38] F. Zhou and G. Cong, "On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization," *arXiv preprint arXiv:1708.01012*, 2017.

[39] B. E. Woodworth, J. Wang, A. Smith, B. McMahan, and N. Srebro, "Graph oracle models, lower bounds, and gaps for parallel stochastic optimization," in *Advances in neural information processing systems*, 2018, pp. 8496–8506.

[40] S. U. Stich, "Local sgd converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.

[41] T. Evgeniou, C. A. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 615–637, 2005.

[42] E. V. Bonilla, F. V. Agakov, and C. K. Williams, "Kernel multi-task learning using task-specific features," in *Artificial Intelligence and Statistics*, 2007, pp. 43–50.

[43] A. Pentina and C. H. Lampert, "Multi-task learning with labeled and unlabeled tasks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2807–2816.

[44] J. Zhou, J. Chen, and J. Ye, "Multi-task learning: Theory, algorithms, and applications," *SDM tutorials*, 2012.

[45] J. Chen, J. Zhou, and J. Ye, "Integrating low-rank and group-sparse structures for robust multi-task learning," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 42–50.

[46] S. Kim and E. P. Xing, "Tree-guided group lasso for multi-task regression with structured sparsity." in *ICML*, vol. 2, 2010, p. 1.

[47] J. Zhou, L. Yuan, J. Liu, and J. Ye, "A multi-task learning formulation for predicting disease progression," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 814–822.

[48] X. Yang, S. Kim, and E. P. Xing, "Heterogeneous multitask learning with joint sparsity constraints," in *Advances in neural information processing systems*, 2009, pp. 2151–2159.

[49] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *Advances in neural information processing systems*, 2007, pp. 41–48.

[50] Y. Zhang and D.-Y. Yeung, "A convex formulation for learning task relationships in multi-task learning," *arXiv preprint arXiv:1203.3536*, 2012.

[51] J. Zhou, J. Chen, and J. Ye, "Clustered multi-task learning via alternating structure optimization," in *Advances in neural information processing systems*, 2011, pp. 702–710.

[52] L. Jacob, J.-p. Vert, and F. R. Bach, "Clustered multi-task learning: A convex formulation," in *Advances in neural information processing systems*, 2009, pp. 745–752.

[53] J. Zhou, J. Chen, and J. Ye, "Malsar: Multi-task learning via structural regularization," *Arizona State University*, vol. 21, 2011.

[54] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 109–117.

[55] Y. Vaizman, N. Weibel, and G. Lanckriet, "Context recognition in-the-wild: Unified model for multi-modal sensors and multi-label classification," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 168, 2018.

[56] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE Pervasive Computing*, vol. 16, no. 4, pp. 62–74, 2017.

[57] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[58] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[59] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville, "Video description generation incorporating spatio-temporal features and a soft-attention mechanism," *arXiv preprint arXiv:1502.08029*, 2015.

[60] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[61] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

[62] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[63] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," *arXiv preprint arXiv:1609.01454*, 2016.

[64] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio, "End-to-end continuous speech recognition using attention-based recurrent nn: First results," *arXiv preprint arXiv:1412.1602*, 2014.

[65] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.

[66] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction." in *IJCAI*, 2018, pp. 3428–3434.

[67] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[68] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *Big Learning NIPS Workshop*, vol. 6, 2013, p. 2.

[69] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[70] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *52nd IEEE conference on decision and control*. IEEE, 2013, pp. 3671–3676.

[71] N. Aybat, Z. Wang, and G. Iyengar, "An asynchronous distributed proximal gradient method for composite convex optimization," in *International Conference on Machine Learning*, 2015, pp. 2454–2462.

[72] Y. Zhang, J. C. Duchi, and M. J. Wainwright, "Communication-efficient algorithms for statistical optimization," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3321–3363, 2013.

[73] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in neural information processing systems*, 2015, pp. 685–693.

[74] V. Smith, S. Forte, C. Ma, M. Takáč, M. I. Jordan, and M. Jaggi, "Cocoa: A general framework for communication-efficient distributed optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 8590–8638, 2017.

[75] 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.

[76] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[77] Y. Zhang and X. Lin, "Disco: Distributed optimization for self-concordant empirical loss," in *International conference on machine learning*, 2015, pp. 362–370.

[78] S. J. Reddi, J. Konečnỳ, P. Richtárik, B. Póczós, and A. Smola, "Aide: Fast and communication efficient distributed optimization," *arXiv preprint arXiv:1608.06879*, 2016.

[79] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[80] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *Journal of Machine Learning Research*, vol. 13, no. Jan, pp. 165–202, 2012.

[81] 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.

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

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

[84] Y. Ruan, X. Zhang, S.-C. Liang, and C. Joe-Wong, "Towards flexible device participation in federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 3403–3411.

[85] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 10, pp. 4229–4238, 2019.

[86] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2019.

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

[88] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.

[89] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3478–3487.

[90] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "Matcha: Speeding up decentralized sgd via matching decomposition sampling," in *2019 Sixth Indian Control Conference (ICC)*. IEEE, 2019, pp. 299–300.

[91] A. Reisizadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, "An exact quantized decentralized gradient descent algorithm," *IEEE Transactions on Signal Processing*, vol. 67, no. 19, pp. 4934–4947, 2019.

[92] O. Dekel, P. M. Long, and Y. Singer, "Online multitask learning," in *International Conference on Computational Learning Theory*. Springer, 2006, pp. 453–467.

[93] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile, "Linear algorithms for online multitask classification," *Journal of Machine Learning Research*, vol. 11, no. Oct, pp. 2901–2934, 2010.

[94] A. Agarwal, A. Rakhlin, and P. Bartlett, "Matrix regularization techniques for online multitask learning," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-138*, 2008.

[95] K. Murugesan, H. Liu, J. Carbonell, and Y. Yang, "Adaptive smoothed online multitask learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4296–4304.

[96] X. Jin, P. Luo, F. Zhuang, J. He, and Q. He, "Collaborating between local and global learning for distributed online multiple tasks," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 113–122.

[97] N. Lu, G. Zhang, and J. Lu, "Concept drift detection via competence models," *Artificial Intelligence*, vol. 209, pp. 11–28, 2014.

[98] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine learning*, vol. 1, no. 3, pp. 317–354, 1986.

[99] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, p. 58, 2004.

[100] N. Lu, J. Lu, G. Zhang, and R. L. De Mantaras, "A concept drift-tolerant case-base editing technique," *Artificial Intelligence*, vol. 230, pp. 108–133, 2016.

[101] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[102] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 377–382.

[103] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 226–235.

[104] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2010, pp. 135–150.

[105] G. Jaber, A. Cornuéjols, and P. Tarroux, "Online learning: Searching for the best forgetting strategy under concept drift," in *International Conference on Neural Information Processing*. Springer, 2013, pp. 400–408.

[106] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, A. Strehl, and S. Vishwanathan, "Hash kernels," in *Artificial intelligence and statistics*. PMLR, 2009, pp. 496–503.

[107] M. Zinkevich, A. J. Smola, and J. Langford, "Slow learners are fast," in *NIPS*, 2009.

[108] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, 2010, pp. 456–464.

[109] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent." in *NIPS*, vol. 4, no. 1. Citeseer, 2010, p. 4.

[110] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 5451–5452.

[111] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker IV, "Efficient large-scale distributed training of conditional maximum entropy models," 2009.

[112] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. Hoi, "On classifying drifting concepts in p2p networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 24–39.

[113] I. Hegedűs, R. Ormándi, and M. Jelasity, "Massively distributed concept drift handling in large networks," *Advances in Complex Systems*, vol. 16, no. 04n05, p. 1350021, 2013.

[114] Y. Vaizman, K. Ellis, G. Lanckriet, and N. Weibel, "Extrasensory app: Data collection in-the-wild with rich user interface to self-report behavior," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 554.

[115] J. M. Jianmo Ni, Larry Muhlstein, "Modeling heart rate and activity data for personalized fitness recommendation," *World Wide Web Conference*, 2019.

[116] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

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

[118] T. Tieleman and G. Hinton, "Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning," *COURSERA Neural Networks Mach. Learn*, 2012.

[119] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[120] R.-G. Cirstea, D.-V. Micu, G.-M. Muresan, C. Guo, and B. Yang, "Correlated time series forecasting using deep neural networks: A summary of results," *arXiv preprint arXiv:1808.09794*, 2018.

[121] Z. Chai, H. Fayyaz, Z. Fayyaz, A. Anwar, Y. Zhou, N. Baracaldo, H. Ludwig, and Y. Cheng, "Towards taming the resource and data heterogeneity in federated learning," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 19–21.

[122] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng, "Tifl: A tier-based federated learning system," *arXiv preprint arXiv:2001.09249*, 2020.

[123] O. Firat, K. Cho, and Y. Bengio, "Multi-way, multilingual neural machine translation with a shared attention mechanism," *arXiv preprint arXiv:1601.01073*, 2016.

[124] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[125] Y.-M. Wang and T. M. Elhag, "On the normalization of interval and fuzzy weights," *Fuzzy sets and systems*, vol. 157, no. 18, pp. 2456–2471, 2006.

[126] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[127] Y. K. Cheung and R. Cole, "Amortized analysis on asynchronous gradient descent," *arXiv preprint arXiv:1412.0159*, 2014.

[128] I. M. Baytas, M. Yan, A. K. Jain, and J. Zhou, "Asynchronous multi-task learning," in *2016 IEEE 16th International Conference on Data Mining*. IEEE, 2016, pp. 11–20.

[129] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "Federated optimization for heterogeneous networks," *arXiv preprint arXiv:1812.06127*, vol. 1, no. 2, p. 3, 2018.

[130] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[131] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[132] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab," in *2019 {USENIX} Annual Technical Conference ({USENIX} {ATC} 19)*, 2019, pp. 1–14.

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

[134] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, 2018.

[135] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*. PMLR, 2014, pp. 1000–1008.

[136] J. Ni, L. Muhlstein, and J. McAuley, "Modeling heart rate and activity data for personalized fitness recommendation," in *The World Wide Web Conference*. ACM, 2019, pp. 1343–1353.

[137] I. Žliobaite and L. I. Kuncheva, "Determining the training window for small sample size classification with concept drift," in *2009 IEEE International Conference on Data Mining Workshops*. IEEE, 2009, pp. 447–452.

[138] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[139] S. Mansukhani, "Data drift detection for image classifiers," *https://blog.dominodatalab.com/data-drift-detection-for-image-classifiers/*, 2019.

[140] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning," in *International Conference on Learning Representations*, 2019.

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

[142] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

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

[144] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N project report, Stanford*, vol. 1, no. 12, p. 2009, 2009.

[145] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

Curriculum Vitae

Yujing Chen received her Bachelor's degree in Digital Media and Art from Beijing University of Posts and Telecommunications, (BUPT), Beijing, China in 2010. She received her Master's degree in Art and Design from Renmin University of China (RUC), Beijing, China in 2016. She joined George Mason University (GMU) in Computer Science Department in 2016, and received a Doctor of Philosophy degree in Computer Science from GMU in 2021. Before joining GMU, she has worked as a designer intern for two years at Intel, Beijing, China in 2013. She also has worked as a Front-end Engineer for one year and nine months at China Software Testing Center, Beijing, China in 2011.