Towards Adaptive Federated Semi-Supervised Learning for Visual Recognition

Min Wen

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science (Quality Systems Engineering) at
Concordia University
Montreal, QC, Canada

August 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Min Wen

Entitled: Towards Adaptive Federated Semi-Supervised Learning for Visual Recognition.

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (**Quality Systems Engineering**)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. J. Bentahar

_____ Examiner
Dr. C. Wang

_____ Examiner
Dr. J. Bentahar

_____ Supervisor
Dr. A. Ben Hamza

Approved by _____
Dr. A. Ben Hamza, Director
Concordia Institute for Information Systems Engineering

_____
Dr. M. Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Date _____

Abstract

Towards Adaptive Federated Semi-Supervised Learning for Visual Recognition

Min Wen

Internet of Things (IoT) devices such as smart phones and wireless sensors have proliferated in smart cities over the past few years. Various applications, including augmented reality, autonomous driving and smart homes, are based on the effective use of data generated by IoT devices. However, two main issues constrain the development of IoT applications. The first issue originates from the data distribution, which is usually isolated and not easy to be centralized due, in large part, to privacy concerns. The second issue arises from the shortage of labeled data, as the labeling process is costly, time-consuming and often requires input from domain experts. More recently, federated semi-supervised learning has become a viable solution to mitigate these issues by collaboratively training a machine learning model using decentralized labeled and unlabeled data. It also brings extensibility and generalizability without privacy infringement compared to traditional centralized training. However, the federated learning process is quite challenging due to data heterogeneity among clients. The contributions in this thesis are two-fold. First, we present an adaptive federated semi-supervised learning framework, which seamlessly integrates adaptive optimizers on both server and client sides in an effort to promote system adaptability. Experiments and ablations studies conducted on four standard benchmark datasets demonstrate the effectiveness of our proposed approach in image classification, achieving superior performance over strong baseline methods.

The other contribution consists of designing a two-stage human activity recognition system, which also incorporates adaptive optimizers into both local and global training. Clients train a local autoencoder model with a learning rate adaptive to local gradients, while the central orchestration server updates the global autoencoder model by applying a gradient-based adaptive optimizer to the average of clients' model updates. Our system leverages a large amount of unlabeled data on clients with the aim of achieving a higher classification accuracy. The key benefit of adaptive optimizers is their ability to improve local training, while stabilizing the global aggregation in a bid to guarantee a proper optimization. We demonstrate through experiments that the proposed framework is robust to non-independent and identically distributed data and yields a stable convergence rate in different settings.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In this chapter, we present the motivation behind this work, followed by the problem statement, objectives of the study, literature review, an overview of federated learning for image classification and human activity recognition, and thesis contributions.

1.1 Framework and Motivation

Federated learning has garnered significant attention in both academia and industry due to its distinctive characteristics of collaboratively training a machine learning model without explicitly sharing data or privacy infringement. Application domains of federated learning include keyboard prediction, wireless networks, financial risk prediction, and medical research [1]. The vast majority of existing federated learning models focus primarily on tackling issues like non-independent and identically distributed (Non-IID) data [2–6] and communicate efficiency [7–12], and typically assume that the training data are fully labeled. However, labeling massive data generated by IoT devices is not always feasible in practice. Therefore, properly utilizing unlabeled data has become a major challenge in federated learning. Recent approaches [13, 14] use unlabeled data on clients, while excluding the possibility of using labeled data locally. Considering both types of data is more realistic, but tends to introduce a severe Non-IID problem. For example, one common solution is to generate pseudo-labels through self-learning for unlabeled data of which distribution is unpredictable. Thus, traditional optimization and aggregation schemes may not be suitable for settings with a large unlabeled data. Other issues such as partial training (*i.e.* only a small fraction of clients participate in each round) and stragglers (*i.e.* clients go offline during training), make it

more urgent to come up with a stable and adaptive method.

1.2 Problem Statement

Image classification and human activity recognition are two ideal application scenarios for distributed learning. Both tasks require large amounts of training data for better model training. This data is usually generated and collected by IoT devices.

1.2.1 Image Classification

Image classification is all about labeling images in a dataset and organizing them into a known number of classes so they can be found quickly and efficiently, and the goal is to assign new images to one of these classes. In supervised learning tasks, the available data for classification is usually split into two disjoint subsets: the training set for learning and the test set for testing. The training and test sets are usually selected by randomly sampling a set of training instances from the available data for learning and using the rest of the instances for testing. The performance of a classifier is then assessed by applying it to test data with known target values and comparing the predicted values with the known values. In semi-supervised learning tasks, a large portion of the training data is unlabeled. Several techniques, such as self-learning and consistency regularization, are used in this context.

1.2.2 Human Activity Recognition

Human activity recognition refers to a classification task where the objective is to identify which human activity is performed by a certain person in a given period. Activities can be of different types such as opening/closing doors, sitting down/standing up, and so forth. Unlike image classification that finds spatial correlations in an image, the objective of human activity is to finding temporal correlation among time series signals. Continuous data is usually partitioned into discrete frames whose features are quantized and then a label is attached in a supervised learning fashion. Deep learning models draw a relation between the sequence of frames, and are thus able to classify time series data. In semi-supervised learning, a common method follows a two-step pipeline by first training an autoencoder using unsupervised learning and then tuning a classifier using supervised learning.

1.3 Objectives

In this thesis, we propose federated semi-supervised learning models for image classification and human activity recognition.

- For image classification, we propose an adaptive federated semi-supervised learning framework for collaboratively training a deep learning model among clients, with the aim to increase classification accuracy, facilitate hyperparameter tuning and make training robust to partial training and stragglers. We simulate the most realistic scenario that clients may have both labeled and unlabeled data, while no data on the central server.
- For human activity recognition, we apply adaptive federated semi-supervised learning to time series data, where we assume the server has a small amount of labeled data while clients hold a large amount of unlabelled data. The objective is to accurately categorize human activity classes for each frame of the test dataset.

1.4 Literature Review

Both image classification and human activity recognition (HAR) are fundamental problems in computer vision, medical imaging, and geometry processing. In our case, we focus on related topics that have already been experimented with in federated learning, semi-supervised learning or adaptive learning. Most federated learning algorithms use image classification accuracy to quantify the model performance. Here, we review general federated learning methods and provide key concepts of semi-supervised learning and adaptive learning. A multitude of HAR systems use different optimizers and are deployed in various networks. Here, we describe HAR applications from different perspectives, including HAR with wearable devices, HAR with federated learning, and HAR with semi-supervised learning.

1.4.1 Federated Learning

The concept of FL was proposed by Google in 2016 [15]. The main purpose of FL is to train a model based on multiple devices without centralizing data. It provides an alternative way to alleviate data shortage in centralized training and protect user privacy. A typical FL has three steps: sever implemented broadcasts global model to participating clients; then clients conduct local training and send back model update to server; finally, server averages model update to update global model. This process repeats until training converges. Afterward, FL is instantiated by *FedAvg* [16] which generates a global model by iteratively computing the average of distributed

model weights instead of gradients of local batch step. This is also the main difference between *FedAvg* and *FedSGD*. Since FL was proposed, it has already triggered numerous applications such as mobile keyboard prediction [17], Human Activity Recognition [18], Recommender System [19], and so on. FL is further surveyed by [20], which categorizes the definition of FL to vertical, horizontal and transfer federated learning. Our work mainly focuses on vertical federated learning and assumes that overall devices' samples constitute the population. Mcmahan *et al.* [16] studied that model in FL is always able to quickly converge among IID datasets in a non-convex problem, while as for unbalanced and Non-IID settings, works [2–4, 6, 21] show that model performance degrades significantly. This is recognized as statistical concern in FL [22, 23]. To resolve this concern, Zhao *et al.* [3] suggested creating a globally shared subset to narrow the accuracy gap. Jeong *et al.* [4] proposed federated augmentation (*FAug*), where each device collectively trains a generative model and thereby augments its local data towards yielding an IID dataset. Peng *et al.* [2] used adversarial domain adaptation to cope with divergence during training.

In addition to statistical concerns, another dominant constraint is the communication overhead. Huge connection execution is required during FL training. For mitigating this problem, Wang *et al.* [24] used atomic decomposition to stochastic gradients sparsification, therefore minimizing the volume of transfer at the same time reducing the variance of gradients among clients. Yao *et al.* [25] proposed two techniques, *i.e.* FedMMD and FedFusion, to reduce the number of communication rounds. FedMMD adds a Maximum Mean Discrepancy loss on local training in case of enlarging divergence between global output and local output. FedFusion makes global and local features fused to achieve higher accuracy with fewer communication costs. Both methods take consideration of outputs from both global and local perspectives in order to speed up convergence. Li *et al.* [26] introduced a proximal term in local loss function to constrain diversity among users. Chang *et al.* [27] considered using multiple access channels to better use bandwidth. Rothchild *et al.* [28] took advantage of momentum and error accumulation from clients to overcome sparse client participation.

Based on these researches, we summarize two major concerns that constrain FL training efficiency:

- Statistical Concerns: Mostly, data on one edge device is drawn from a different distribution, *i.e.* Non-IID and the number of data in each device may be unbalanced.
- Effective Concerns: The ability of Edge Network depends heavily on network bandwidth, edge computing resources (CPU, GPU, storage), and local battery level. Together with statistical concerns, these effective challenges make issues such as stragglers and fault tolerance significantly more prevalent than in a typical data-centered environment [22].

1.4.2 Semi-supervised Learning

Semi-supervised learning (SSL) is halfway between supervised and unsupervised learning. It makes use of unlabeled data in an unsupervised way while still conducting supervised training [29]. Many recent works [30–33] have shown SSL strong stability in dealing with a small amount of labeled data in image classification. In our work, we utilize the proxy-label technique to generate pseudo-label for unlabeled data. Along with consistency regularization, we define two types of loss, *i.e.* classification loss and consistency loss, in semi-supervised learning. Referring to [32], *consistency regularization* is an important component of many recent state-of-the-art semi-supervised learning algorithms. It was first proposed as regularization with stochastic transformation and perturbation for deep semi-supervised learning in work [34]. Then different techniques (*e.g.* data augmentation [35], stochastic regularization [34] and adversarial perturbations [36]) are adopted to perturb consistency. Dai *et al.* [37] even show that a heavier perturbation would produce better results. *Pseudo-labeling* is one type of method in self-training, which was first empirically emerged in Mclachlan's work [38] and has been widely accepted as a basic self-labeling method in many fields, including object detection, image classification, and name a few. In specific, pseudo-labeling represents a labeling process that generates a hard label based on meeting certain criteria. Recent works [36, 39, 40] demonstrate that pseudo-labeling is a powerful component in semi-supervised learning as it can be adapted into entropy minimization [31].

1.4.3 Federated Semi-Supervised Learning

Many works [13, 14, 41–43] argue that implementing SSL in FL is non-trivial. According to a unique characteristic (*i.e.* decentralized data) of FL, Jin *et al.* [44] divided federated semi-supervised learning (FedSSL) into two streams: label-centralized FedSSL and label-distributed FedSSL. As the names suggest, two categorizations differ at the location of labeled data. Moreover, many works [43, 45, 46] studied FL where sample features are non-unique among users, which is different from our aspects that samples hold share same feature spaces. This can be seen as another categorizing criterion that is beyond our scope.

Labeled-centralized FedSSL allows central server to participate in training. As server always stores official labels (compared with user-defined labels), it conducts supervised training concurrently with local training [13] or sequentially [41] after local training. However, the number of labeled samples is always limited. *Labeled-distributed* FedSSL utilizes enormous unlabeled data generated by users on which local representation or logit is first produced. Itahara *et al.* [42] shared labels instead of weight update among clients to distill labels for unlabeled samples, and also limited communication cost scaling up when model size increases by only updating predicted

logits instead of model parameters. One of the main defects of this is that local labels are imprecise, resulting from clients heterogeneity. Jeong *et al.* [14] considered both distributed scenarios and introduce inter-client consistency loss by exchanging outcomes between users. This approach diminishes the heterogeneity effect among devices at the cost of inefficient communication.

1.4.4 Adaptive Federated learning

Adaptive optimizers (*e.g.* AdaGrad, AdaDELTA, ADAM) have been theoretically and empirically studied that they are convergence-guaranteed for solving non-convex problems. Many works [16, 47, 48] have fully studied the convergence performance with non-adaptive optimizations (*e.g.* SGD) in a distributed setting. While hardly works use distributed adaptive method, Xie *et al.* [49] adopted AdaGrad as local optimizer to add adaptivity on user side. Reddi *et al.* [50] proposed that adaptive optimizer on server can halve communication cost and local memory usage while remaining comparable training performance. Instead of using AdaGrad, Tong [51] studied ADAM and AMSGrad in FL and provide a scheme of calibration for adaptive learning rate. The main motivation of our work comes from a variant of the combination of *FedAdaGrad* [50] and *AdaAlter* [49].

1.4.5 HAR with Wearables

Human movement data recorded with sensors on the body are multivariate time-series data with characteristics of high spatial and temporal resolution [52]. Bulling *et al.* [53] provided a baseline tutorial to analyze this type of data from body-worn sensors. The first step is partitioning the sequential data into several continuous segments by sliding windows. These segments can be partially overlapped or distinctive due to the method used for partition. The second step is to extract the features of each segment. Typical features such as statistical features [54], basis transform features [55], multi-level features [55] or bio-mechanical features [56] are drawn and form a newly discrete dataset. After that, a classifier such as support vector machine, decision tree or graph model is trained in a typical machine learning way. This process always needs hand-crafted feature extraction and thus heavily relies on expertise in certain domains.

Apart from manually selected features, deep learning has emerged as an alternative to automatically extract features without the need for time-consuming and costly feature engineering procedures. Plotz [57] employed autoencoder-based feature learning on sequential data and showed that deep autoencoder outperforms traditional statistical feature extraction. Besides, many works choose convolutional neural networks (CNNs) as the feature extraction model [58, 59] or an end-to-end classifier [60]. Yang *et al.* [59] fed raw sensory data into a convolutional network to extract

discriminative features following two dense layers as the classifier. Yao *et al.* [60] trained a fully convolutional neural network to classify sequences of sensory data directly. This avoids multi-class windows problem caused by sliding window. We normally treat a single frame from sensory data as an individual and independent activity. However, we intuitively deem that there is a temporal dependence between contiguous frames. Many works find deep recurrent neural network works better in classifying sequential data. Ordonez *et al.* [58] proposed a generic deep framework for activity recognition based on convolutional and long short-term memory (LSTM) recurrent units. They used a 5-layer 1-D convolutional network to construct features and took a 2-layer LSTM network as classifier. After many works evolve, Hammerla *et al.* [52] studied the relative effect of model types, architectures or training techniques to model performance and provides directions towards parameter tuning through thousands of experiments.

1.4.6 Federated Learning for HAR

Thanks to the advancement of computational ability at edge devices, HAR is one of the well-motivated subjects that can benefit from FL. According to the distribution of feature space of training data, Yang *et al.* [20] categorized FL into three types: horizontal, vertical and transfer FL. Most federated HARs [18,41,61,62] belong to horizontal FL that all clients share the same feature spaces while having different sample spaces. Due to this characteristic, individual devices such as *i.e.* mobiles or wearables can collect samples and train a global model in a distributed way. Most federated HAR works are solving three problems: personalization, application and data scarcity. In general, Konstanin *et al.* [18] studied the new hybrid method for HAR that combines semi-supervised and federated learning to take advantage of both approaches and evaluated model performance in FL with finding acceptable accuracy compared with training in centralized learning. Qiong *et al.* [63] proposed ProFit which integrates personalized federated training that includes federated meta-learning [64], federated multi-task learning [22], federated distillation [4, 65] and data augmentation [4] into the basic FL framework. Chen *et al.* [61] studied adding local transfer learning besides FL can largely promote personalized recognition performance. Apart from personalization, Feng *et al.* [66] gave a privacy-preserving mobility prediction framework without personal information infringement. Works [61, 67, 68] provide specific application scenarios that realize monitoring health conditions with Federated HAR. In specific, Federated HAR applications are required to protect users' privacy and recognize patterns in time and accurately.

Another trendy field in federated HAR is solving the lack of labeled data. Most works assume that data acquired from local devices are labeled, and supervised learning is thus pervasively adopted. This scenario is hard to achieve as labeling time-series data, unlike labeling image data,

is more error-pruning and time-consuming. Moreover, pre-processing sensory data from wearable needs expertise and is objective-oriented. Works [41, 41, 62] sought to utilize semi-supervised learning to automatically labeling or decoding data. We compare and detail these works in Section 1.4.7.

1.4.7 Semi-supervised Learning for HAR

Semi-supervised learning takes both labeled and unlabeled data as input when training machine learning models. It makes training models without sufficient labeled data produce considerable improvement in learning accuracy. Several semi-supervised learning methods such as pseudo-labeling, transfer learning and autoencoder have brought advantages in centralized training. In HAR, annotating time-series data needs expertise and is time-consuming, intrusive, and costly. Thus, using semi-supervised learning to tackle data scarcity is becoming more popular in this domain. In computer vision or text classification, many semi-supervised methods produce proxy-label as ground truth for unlabeled data. This is based on that these two types of data can have different forms of augmentation and perturbation that are used to train models adversarially. For time-series data, data augmentation is typically used to make synthetic data to complement original data [69–72]. Speaking of generating proxy-label, approaches including self-learning [73], co-learning [74] and multi-graph learning [75] choose the most confident label predicted by model or directly comparing the similarity between unlabeled data and labeled data. Apart from generating pseudo-label, active training is another way that needs users to input labels towards unlabeled data actively. BETTINI *et al.* [41] used both active learning and label propagation to annotate local streams of unlabeled sensor data semi-automatically.

Autoencoder is an alternative operation making unlabeled data available in HAR. Instead of making pseudo-label, it learns how to extract useful features for further utilization. Some works use it to extract representative features for understanding raw data [76] or reduce computation overhead [77]. Vincint *et al.* [76] trained adversarial autoencoders to precisely reduce data dimensionality for the purpose of visualizing high dimensional time-series data. Bandar *et al.* [77] used stacked autoencoder (SAE) classifier which trains classifier and autoencoder simultaneously. Its results show that SAE classifier can enhance the recognition accuracy and decrease recognition time that suits poor-computation devices. In addition to this, most works are following a two-step pipeline. First, an autoencoder is trained to learn latent representations from labeled or unlabeled data in unsupervised learning. This process is irrelevant to the task. Second, learned representations of labeled data are extracted and used for training classifier. Varamin *et al.* [78] conducted unsupervised feature learning using a convolutional auto-encoder prior to supervised

learning. This largely mobilizes sufficient unlabeled data and helps extract useful high-level features of labeled data. Similarly, Gu *et al.* [79] investigated the application of stacked denoising auto-encoder for automatic feature extraction. Denoising auto-encoder is an extension of a simple autoencoder that trains autoencoder by input with noise while still reconstruct input without noise. It is proved effective in building more robust and relevant features for further classification. Gao *et al.* [80] combined stacked denoising auto-encoder with LightGBM, a supervised classification method that reveals the inherent feature dependencies among categories for accurate human activity recognition. Wei *et al.* [81] used characteristics of traffic flow data extracted by autoencoder and a Long Short-Term Memory(LSTM) to predict complex linear traffic flow data. Han et al. [82] proposed an Autoencoder Long-term Recurrent Convolutional Network that uses autoencoder to sanitize the noise in raw data following a convolutional neural network module to extract high-level representative features. Li *et al.* [83] conducted feature extraction in channel-wise instead of in a monolithic way. It combines the features extracted from each channel and then encodes the feature by three methods, *i.e.* sparse auto-encoder, denoising auto-encoder and Principal component analysis. From these points of view, we can easily adapt this two-step framework into FL of which server trains classifier and client trains autoencoder. Zhao *et al.* [41] applied this pipeline in an FL setting where edge devices implement unsupervised training while central server conduct supervised training. Our work follows the same framework as work in [41] while adding adaptive learning to increase learning accuracy and robustness.

1.5 Overview and Contributions

The organization of this thesis is as follows:

- Chapter 1 begins with the motivations and goals for this research, followed by the problem statement, the objective of this study, a literature review with a brief discussion of some algorithms relevant to federated learning in image classification and human activity recognition.
- In Chapter 2, we introduce an adaptive federated semi-supervised learning model, which employs an adaptive optimizer on both local and server sides. Our experimental results demonstrate its effectiveness of our approach on visual classification. Our method is more robust than other federated semi-supervised methods in the sense that we consider a more realistic scenario. Our framework is compatible and resilient with three types of clients: partial participation, stragglers and unbalanced data. Our framework is also easy to tune with a small amount of interplay among key hyperparameters, and achieves superior performance over other standard FedSSL or adaptive federated methods.

- In Chapter 3, we apply adaptive federated semi-supervised learning to human activity recognition. Our method has a better mean F1-score compared with other baseline methods, and is robust to partial participation and Non-IID data. Our ablation studies demonstrate the effectiveness brought by the key elements of our framework. We provide insight into data partition and partial updating, which play an important role in an effective federated framework.
- Chapter 4 presents a summary of the contributions of this thesis, limitations, and outlines several directions for future research in this area of study.

CHAPTER 2

Semi-Supervised Image Classification

In this chapter, we introduce an adaptive federated semi-supervised learning that exploits an adaptive optimizer on both local and server sides. The training process consists of two main steps, namely local update and global aggregation. More specifically, our proposed solution is geared towards labeled-distributed horizontal federated learning problem, where data among clients shares the same feature spaces. We assume that a curious and trustworthy server will not host data, but deals only with weights aggregation. During local updates, clients perform multiple epochs of training using local adaptive optimizers on their local data, where unlabeled data are labeled via pseudo-labeling through consistency regularization. Clients transfer model updates, the number of training samples and local gradient accumulators back to server, where a gradient-based server optimizer updates the global model to the average of clients' updates. The central server broadcasts the new global model and averaged accumulator to clients selected in the next round. In this framework, we consider a more realistic scenario that edge users may have intricate labeling behaviors, and that the number of participated devices varies at each round to simulate stragglers. Our framework is easy to tune with a small interplay among key hyperparameters and achieves superior performance compared to standard federated semi-supervised or adaptive federated methods.

2.1 Introduction

Smart Cities aim to solve common urban challenges such as energy consumption, traffic congestion, environmental pollution or achieving intelligent scenes such as automatic human activity recognition in the Internet of Things (IoT) system [84]. Numerous IoT applications such as aug-

mented reality, autonomous driving, surveillance and industry generate a significant amount of data that largely exceeds today's computational capability [85]. Other IoT devices such as motion sensors, ambient sensors, or thermostats collect various time series data that require efficient data processing. Traditional machine learning methods use centralized data stored at a data center and require data transfer from a massive number of distributed IoT devices to a third-party location. However, this learning approach is no longer sustainable due to several concerns. The first concern arises from the unprecedented volume of data. Massive data transmission to a centralized center involves long propagation delays and incurs unacceptable latency [86]. Despite a delayed reaction, centralized processes can burden the backbone networks, especially in tasks involving unstructured data [87]. Another concern is that servers are unable to properly exploit massive data. On the one hand, data in Smart Cities is sensitive; and sending data to a centralized location may cause privacy infringement. Also, data transfer can be inefficient as resource starvation may raise network congestion. This is because many users endeavor to make use of the same resource at the same time if the system requires an in-time update [88]. To this end, federated learning (FL) was proposed to help alleviate such challenges by allowing multiple IoT devices to collaboratively build machine learning models without explicitly sharing data [44].

Most existing FL frameworks assume that aggregating each device's data will lead to a complete dataset. This assumption is mostly unrealistic, as available data of one device is always imbalanced and is collected as a highly-skewed distribution. This is referred to as the statistical concerns in federated learning. Additionally, in many fields such as medical, on-device data is largely or totally unlabeled because labeling is very costly and requires domain expertise. These make solely supervised federated learning impractical. Semi-supervised learning (SSL) [29] is one of the widely adopted paradigms utilizing unlabeled data to enlarge the scale of the training dataset. It takes both labeled and unlabeled data as input when training machine learning models. It falls between unsupervised learning and supervised learning, and makes training models without sufficient labeled data to produce considerable improvement in learning accuracy. Several semi-supervised learning methods such as proxy labeling, transfer learning and autoencoder have brought advantages in centralized training. Intuitively, incorporating semi-supervised learning into FL would be a non-trivial solution. Until now, federated semi-supervised learning (FedSSL) has been tested as a practical way to tackle the above issues [13, 14, 41, 44, 62]. FedSSL typically uses multi-view training (i.e. generating pseudo-labels through different views of unlabeled data) technique to label unlabeled data. Nevertheless, statistical concerns still remain and worsen in FedSSL because unlabeled data could bring unprecedented data heterogeneity [1]. Apart from this, most FL works set partial participation training (i.e. selecting a subset of devices at every training round) as a primary setting

that may cause deviation from global optima to local optima due to the heterogeneity introduced by device selection. For example, the number of steps for initializing a step-based learning rate schedule is thus unclear as it is dependent on clients' training frequency, which becomes a random number in partial training. In this case, a centralized optimization schedule (e.g. uniform learning rate) is no longer proper in FedSSL.

Another essential bottleneck in FL is related to communication efficiency. Traditional distributed optimization methods such as distributed stochastic gradient descent (SGD) incur heavy cost when transmitting data between server and clients. Mcmahan *et al.* proposed *FedAvg* [16], attempting to mitigate this issue by increasing updates at edge devices before communication. However, this approach lacks adaptivity (the ability to incorporate past learning knowledge) and performs worse when training on a continuous data stream. Several approaches [1, 49, 50] add adaptive optimizers to either server and client or make selective updates at each training round. These approaches, however, introduce a multitude of hyperparameters that may cause intrusive influence to original training at the same time. For instance, the level of adaptivity on server and clients may interplay and negatively affect hyperparameter tuning. Therefore, figuring out the relationship between hyperparameters is an essential process for tuning FedSSL models.

Based on the above challenges, we summarize the three main challenges in adaptive FedSSL as follows:

- **Statistical challenge.** Most works study the effect of original data heterogeneity among local devices, while neglecting heterogeneity introduced by semi-supervised learning. For image data, for instance, this skewed distribution typically arises from the pseudo-labeling process, as the local data distribution may be considerably distinctive after labeling. Moreover, the accuracy of labels may be related to the model performance. If the global model converges to local optima at an early stage, labeling would be in vain for afterward training. This also increases sensitivity to the threshold, which is set in pseudo-labeling to manage labeling confidence. All unlabeled samples on a device may be invalid for not being above the threshold.

- **Optimization challenge.** Most FedSSL frameworks use a generic optimizer, such as a centralized SGD or AdaGrad managed by a server. At each training round, server broadcasts updated learning rate and other information to devices. However, this is improper with partial participation if we apply a general learning rate schedule. For example, the learning rate is close to zero at the end of an annealing schedule, while the model does not thoroughly study some local features due to late participation. In a Non-IID setting, global optima may be easily led to local optima due to a large learning rate at the beginning. Moreover, some learning

rate schedules cannot be properly initialized without knowing beforehand information such as the number of steps, which require a concrete number of epochs. Hyperparameter tuning is another issue. With learning taking place on both server and client, more attention should be paid to their interaction effect. For example, a higher global learning rate may have an adverse effect on choosing the best local learning rate. The level of optimal global and local adaptivity rates may vary with different datasets.

- **Generalization challenge.** Most FedSSL systems only consider one type of edge devices in terms of the type of training sample (i.e. all clients are solely labeled or unlabeled). This is unrealistic in today's IoT devices, in which samples may be totally labeled, partially labeled, or totally unlabeled. In order to increase generality, exceptional circumstances such as straggler or partial participation are not taken into account, while these are common in the actual scenario. Moreover, making the framework applicable for different types of data is intractable. Dense data and sparse data comply with diverse settings.

From these points of view, we introduce a realistic and robust adaptive federated semi-supervised learning (AdaFedSSL) framework. For image data, we propose a solution specifically geared towards labeled-distributed horizontal federated learning problems and scenarios of which edge users may have complex labeling behaviors. We consider that the number of participated devices varies at each round for both settings to simulate stragglers. In summary, our main contributions are as follows:

- We introduce AdaFedSSL, which adopts an adaptive optimizer on both local and server sides. Our experiments demonstrate the effectiveness in visual classification.
- Our method is more robust than other FedSSL methods as we consider a more realistic scenario. Our framework is also resilient with three different clients, partial participation, stragglers, and imbalanced data. Moreover, it is easy to tune with a small amount of interplay among key hyperparameters.
- Our extensive experimental results show AdaFedSSL produces superior performance over standard FedSSL or adaptive federated methods, achieving higher test accuracy compared with the baselines.

Figure 2.1 shows our proposed framework. The main distinction among other works is that we use adaptive optimization, *i.e.*, *AdaGrad* [89], at two locations, *i.e.*, client (local) side and server (global) side. There are three types of clients. Labeled clients only contain labeled training

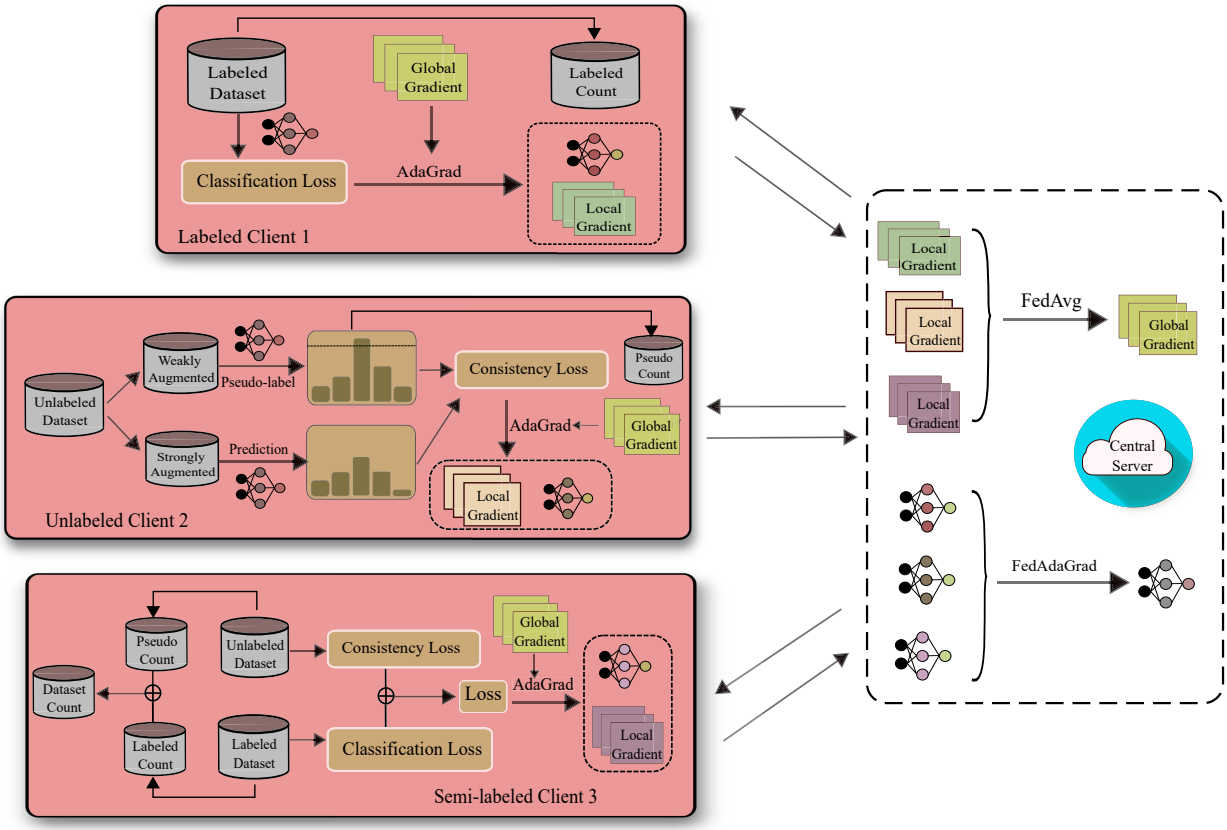

Figure 2.1: Proposed framework for adaptive federated semi-supervised learning.

images. Unlabeled clients only contain unlabeled data that the number of pictures labeled by FixMatch is treated as the weighting of *FedAvg*. Semi-labeled clients have both types of data of which the weighting counts labeled data after pseudo-labeling. Each round, clients transfer local gradient, local parameter and weighting to server where *FedAvg* and *FedAdaGrad* take place. This framework brings better training performance significantly towards four baseline tasks. Detailed information about our framework is described in Section 2.3.

The rest of this chapter is organized as follows. In Section 2.2, we provide preliminaries on adaptive federated learning and federated semi-supervised learning, followed by a problem formulation. In Section 2.3, we introduced our proposed framework, and describe the main algorithmic steps. In Section 2.4, we present extensive experiments to demonstrate the competitive performance of our approach on four standard benchmark image datasets.

2.2 Preliminaries and Problem Statement

Notation For $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, we let $\sqrt{\mathbf{a}}, \mathbf{a}^2$ and $\frac{\mathbf{a}}{\mathbf{b}}$ denote the element-wise square root, square, and division of the vectors. The terms *user*, *client* and *device* are used interchangeably.

2.2.1 Preliminaries

Federated Learning is a distributed learning paradigm of reaching a global objective by separately achieving local objectives. For a traditional FL, the global objective is formulated as:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} F(\boldsymbol{\theta}) = \frac{1}{K} \sum_{i=1}^{K} f_i(\boldsymbol{\theta}) \tag{2.1}$$

where K is the number of training clients and $f_i(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z} \sim \mathcal{D}_i}[\ell(\boldsymbol{\theta}; \mathbf{z})]$ is the loss function calculated by feeding forward batched data $\mathbf{z}$, sampled from training data of i -th clients $\mathcal{D}_i$, to model with parameter $\boldsymbol{\theta}$. A canonical FL consists of two phrases: *local update* and *global aggregation*.

- *local update* denotes local training process on clients. At each round, participating clients will receive global model parameter and replicate it as a local model, which is then be updated on local data.
- *global aggregation* denotes the averaging process on server. Server receives all local model parameters and combines them as one updated global model.

A multitude of training techniques, including partial training and multiple local epochs, can be applied to reduce the high communication cost of transmitting parameters between server and clients. For example, serve can select a small fraction of clients to conduct multi-epoch local updates at each round. Besides, both phrases have chances to adjust objective solvers, such as stochastic gradient descent (SGD) or adaptive optimizers, according to different tasks. FL assumes local loss minimization leads to global loss minimization. Thus, to achieve this global objective, Mcmahan *et al.* proposed *FedAvg* [16] that takes SGD as the local solver. At each round, server selects a subset from total clients and broadcasts global model parameters to local clients. Clients run SGD for several epochs and obtain updated local parameters. Supposing at each round, server selects S number of clients and broadcasts model parameters $\boldsymbol{\theta}_g$ to local clients as $\{\boldsymbol{\theta}_i\}, i \in (1, 2, ..., S)$. Partially selected clients independently update local parameters with local dataset $\mathcal{D}_i$ by gradient descent $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \nabla f_i(\boldsymbol{\theta}_i)$. Global server aggregates each trained model update and takes the average as a new global model, *i.e.*, $\boldsymbol{\theta}_g = \frac{1}{S} \sum_{i=1}^{S} \boldsymbol{\theta}_i$. The newly updated global parameters are then broadcast to clients chosen at the next round. The whole process repeats until convergence.

The amount of local training is controlled by the number of local epochs and the fraction of clients that can be tuned to reduce communication costs.

Data distribution, whether on server or on clients, is typically categorized into independent and identically distributed (IID) or Non-IID. In an IID setting, all clients have the same data distribution, *i.e.*, each client has the same number of classes and the same number of samples with other clients. For a Non-IID setting, clients have various distributions on either the number of classes or the number of samples at each class, *i.e.*, $\mathcal{D}_i \neq \mathcal{D}_j$ or $n_i \neq n_j, \forall i \neq j$, where n_i, n_j represent the number of samples in $\mathcal{D}_i, \mathcal{D}_j$, respectively. We see data as balanced or unbalanced according to the number of samples among clients. *FedAvg* weights local model update by the number of samples of each client. In this regard, server aggregation can be rewritten as $\boldsymbol{\theta}_g = \sum_{i=1}^{S} \frac{n_i}{n} \boldsymbol{\theta}_i$, where $n = \sum_i^S n_i$ is the total number of training samples from all clients.

Adaptive federated learning integrates adaptive optimizer into FL. In our work, AdaGrad [89] is chosen for adapting learning rate to past accumulated parameters. Supposing at global round t, we have partial clients with local models $\{\boldsymbol{\theta}_i^t\}_{i=1}^S$ and a unified local learning rate η_l. We can write the i-th local update as:

$$\boldsymbol{\theta}_i^{t+1} = \boldsymbol{\theta}_i^t - \eta_l \nabla H(\boldsymbol{\theta}_i^t, \mathcal{D}_i) \tag{2.2}$$

where $H(\boldsymbol{\theta}_i^t, \mathcal{D}_i)$ is the loss function with input parameter $\boldsymbol{\theta}_i^t$ and i-th local dataset $\mathcal{D}_i$. At each local iteration, i-th local client samples $\mathbf{z}$ from $\mathcal{D}_i$ and compute gradient $\mathbf{g} = \nabla H(\boldsymbol{\theta}_i^t, \mathbf{z})$. We can make an alternation to this setting to make local learning adaptive to local gradients. For example, we can use AdaGrad instead of SGD on each client. Xie *et al.* [49] proposed a SGD variant based on AdaGrad to perform multiple epochs of training on local datasets. Instead of using a generic learning rate, local learning rate η_l adapts the extent of learning to the frequency of occurring parameters. In particular, a parameter with frequently occurring features will have a small update and vice versa. It introduces a client accumulator to record past gradients, which bring overhead on clients storage to quantify the frequency. This is tested effectively on non-convex problems while introducing extra communication burden. Supposing at local iteration e, let $\mathbf{g}_i^e = \nabla H(\boldsymbol{\theta}_i^e, \mathbf{z}^e), \mathbf{z}^e \sim \mathcal{D}_i$ denotes the stochastic gradient of i-th model; $(\mathbf{v}_i^e)^2 = \sum_{b=1}^{e} \mathbf{g}_i^b \circ \mathbf{g}_i^b$ refers to the accumulator of the squares of gradient up to local iteration e. Note $\mathbf{g}_i^e$ is gradient matrix with the same size as $\boldsymbol{\theta}_i \in \mathbb{R}^d$ and $\circ$ represents coordinate-wise product. We can define the adaptive local update as:

$$\boldsymbol{\theta}_i^{e+1} = \boldsymbol{\theta}_i^e - \eta_l \frac{\mathbf{g}_i^e}{\sqrt{(\mathbf{v}_i^e)^2 + \tau_l^2}} \tag{2.3}$$

where τ_l is a smoothing term for avoiding division by zero while determining the extent of adaptivity at the local side. At the synchronization period, server averages, not only model parameters but

also the client accumulators. This guarantees the same learning rate among participating clients at the beginning of each round.

In addition to local adaptive learning, many works study the effectiveness of server-side adaptive optimizers [50] or momentum [6]. Let server has global parameter $\boldsymbol{\theta}_g^t$ at round t, we can redefine the *FedAvg*'s server update as:

$$\boldsymbol{\theta}_g^{t+1} = \boldsymbol{\theta}_g^t - \frac{1}{S}\sum_{i=1}^{S}(\boldsymbol{\theta}_g^t - \boldsymbol{\theta}_i^{t+1}) \tag{2.4}$$

If we rewrite $\Delta_i^t := \boldsymbol{\theta}_g^t - \boldsymbol{\theta}_i^{t+1}$ and $\Delta^t := \frac{1}{S}\sum_{i=1}^{S}\Delta_i^t$, the server update is equivalent to applying SGD to gradient Δ^t with learning rate $\eta_g = 1$. This makes the foundation of applying an adaptive learning schedule on server side. Let $\mathbf{w}^t = \sum_{b=1}^{t}(\Delta^b)^2 \circ (\Delta^b)^2$ as the server accumulator of past gradient square up to round t, we can rewrite adaptive server update as:

$$\boldsymbol{\theta}_g^{t+1} = \boldsymbol{\theta}_g^t - \eta_g \frac{\Delta^t}{\sqrt{\mathbf{w}^t + \tau_g^2}} \tag{2.5}$$

where τ_g is also a scalar hyper-parameter that avoids zero division and controls the adaptivity on server aggregation.

Federated Semi-Supervised Learning is applying federated semi-supervised learning on traditional machine learning models. Let us define a client with B number of labeled examples $\mathcal{X} = \{(\mathbf{x}_b, \mathbf{p}_b) : b \in (1, ..., B)\}$ where $\mathbf{x}_b$ is a training example and $\mathbf{p}_b$ is its corresponding one-hot label. The client has U number of unlabeled examples $\mathcal{U} = \{\mathbf{u}_b : n \in (1, ..., U)\}$ where $\mathbf{u}_b$ is an unlabeled training vector. Let $p_{\boldsymbol{\theta}}(y|x)$ be the predicted class distribution through a model with parameter $\boldsymbol{\theta}$ for an input x where $p_{\boldsymbol{\theta}}$ is a stochastic function. We define $H(\mathbf{p}, \mathbf{q})$ as the cross-entropy between probability distribution $\mathbf{p}$ and $\mathbf{q}$. FixMatch [32] uses two types of augmentations: strong and weak that denoted by $\mathcal{A}(\cdot)$ and $\alpha(\cdot)$, respectively. For an image classification task, **consistency regularization** assumes that model should output similar outcomes given weak and strong augmented features. Thus for labeled dataset $\mathcal{X}$, predicted distribution from original input and weakly augmented input should be similar. We can write the supervised cross-entropy loss for labeled examples as:

$$\ell_s = \frac{1}{B}\sum_{b=1}^{B} H(\mathbf{p}_b, p_{\boldsymbol{\theta}}(y|\alpha(\mathbf{x}_b))) \tag{2.6}$$

Pseudo-Labeling makes artificial labels for unlabeled data with a constraint that only retaining the labels whose largest class probability surpasses a predefined threshold. Let $\mathbf{q}_b = p_{\boldsymbol{\theta}}(y|\alpha(\mathbf{u}_b))$ denote the predicted class distribution of a given unlabeled example $\mathbf{u}_b$. We take $\hat{\mathbf{q}}_b = argmax(\mathbf{q}_b)$ as corresponding pseudo-label. Note that we assume $argmax$ function produce a "one-hot" probability distribution uniform with labeled example. With consistency regularisation, we can define

the unsupervised cross-entropy loss for unlabeled examples as:

$$\ell_u = \frac{1}{U}\sum_{b=1}^{U} \mathbb{1}(\max(\mathbf{q}_b)) \geq \hat{\tau}) H(\hat{\mathbf{q}}_b, p_{\boldsymbol{\theta}}(y|\mathcal{A}(\mathbf{u}_b))) \tag{2.7}$$

where the threshold $\hat{\tau}$ is a scalar hyperparameter that controls the confidence we retain pseudo-label. Supposing a federated network has K number of clients where each client has labeled data and unlabeled data, namely $\mathcal{D}_i = \mathcal{X}_i \cup \mathcal{U}_i, i \in (1, 2, ...K)$. Let $\ell_s(\mathbf{x})$ and $\ell_u(\mathbf{u})$ denote the loss term for labeled sample $\mathbf{x}$ and unlabeled sample u, respectively. Based on Equation 2.1, we denote the objective of federated semi-supervised learning as:

$$\min_{\theta \in \mathbb{R}^d} f(\theta) = \frac{1}{K}\sum_{i=1}^{K} \mathbb{E}_{(\mathbf{x},\mathbf{u})\sim(\mathcal{X}_i,\mathcal{U}_i)}[\ell_s(\mathbf{x}) + \lambda \ell_u(\mathbf{u})] \tag{2.8}$$

where λ is a fixed scalar hyperparameter representing the relative weight of unlabeled data to the labeled data.

2.2.2 Problem Formulation

The traditional solution of federated semi-supervised learning has two main issues. First, it is improper to simply average local objectives with the same weight as a global objective in an unbalanced setting. Local update (Equation 2.2) aims to grasp a high-level representation of local data samples. However, the level of representation partially depends on the number of samples. For example, local representation learned may be highly skewed towards the class of the largest amount; thus, the averaged global objective would deviate. Considering pseudo-labeling brings worse heterogeneity, it is crucial to take weighted aggregation as server update instead of simple average. For weighted *FedAvg*, the global objective is:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} f(\boldsymbol{\theta}) = \sum_{i=1}^{K} p_i f_i(\boldsymbol{\theta}) \tag{2.9}$$

where p_i is the client's weight where $\sum_{i=1}^{K} p_i = 1$. Let $\mathcal{D}_i = \mathcal{X}_i \cup \mathcal{U}_i$ denotes the set of data samples of i-th client and $B_i = |\mathcal{X}_i|$ and $U_i = |\mathcal{X}_i|$ denotes the number of labeled samples and unlabeled samples. *FedAvg* regards the ratio of client's data samples to overall data samples in an FL system as the client weight, *i.e.*, $p_i = \frac{B_i+U_i}{\sum_{a=1}^{K} B_a+U_a}$. Here instead, for the number of unlabeled data, we only count the samples whose predicted probability of weak augmented sample surpasses predefined threshold:

$$U_i = |\mathbb{1}(\max(p_{\boldsymbol{\theta}}(y|\alpha(\mathbf{u}))) > \hat{\tau})|, \mathbf{u} \sim \mathcal{U}_i \tag{2.10}$$

where $p_{\boldsymbol{\theta}}(y|\alpha(\mathbf{u}))$ represent predicted class distribution of input $\alpha(u)$ with function parameter $\boldsymbol{\theta}$. Let $n_i = B_i + U_i$, we define the weighting of i-th client as:

$$p_i = \frac{n_i}{\sum_{i=1}^{K} n_i} \tag{2.11}$$

Data imbalance caused by pseudo-labeling is partially mitigated by this end.

The second issue relates to the skewed data distribution, *i.e.*, statistical concerns in FL. In a Non-IID setting, global objective hardly converges due to data heterogeneity. This is worsened by partial participation that stochastic clients selected at each round introduce client drift. For example, a single client may be selected several times while others only attend one or zero training round. Moreover, late training may be insufficient if learning rate decay or other annealing schedules applies. In this case, data features may not be aggregated if clients are selected at the end of the schedule. Instead of using a generic optimizer, we use an adaptive optimizer on both local and server sides. Clients inherit local learning rate η_l and apply adaptive gradient descent based on their own gradient record of Equation 2.3. Server in the same way stores global gradient records. This ensures that local representation effectively generalizes local features regardless of the order of training. Through our experiments, we find that global objective tends to diverge in lower communication frequency or higher Non-IIDness. Reddi *et al.* [50] empirically and theoretically proved that an adaptive optimizer of Equation 2.4 on server side is effective and robust for heterogeneity. Our method combines the client accumulator and server accumulator and thus adds adaptivity on both sides. Note that local client accumulator is averaged with server aggregation and broadcast to corresponding participated clients. This means the communication cost is doubled due to communicating accumulator which shares the same size with model parameters. Besides, a scalar number of trained examples for calculating p_i is also sent from client to server, of which communication cost is negligible compared to the size of parameters.

Another prominent issue related to federated semi-supervised learning is the robustness, which normally consists of the tolerance to stragglers, and the ease of hyperparameters tuning. Stragglers or dropped-out clients indicated that the number of selected clients before local training differs from the number of clients sending model updates to the server. Our goal is to testify that the model performance is not significantly affected by stragglers. Adaptive optimizing introduces momentum parameter τ, which is referred to as adaptivity rate in our thesis. While applying an adaptive optimizer on clients and server sides, the number of parameters doubles and makes hyperparameter tuning difficult. Together with tuning two learning rates, learning process is made harder to control. Our goal is to demonstrate that the performance of AdaFedSSL is robust with tuning learning rates and adaptivity rates.

2.3 Proposed Framework

We simulate AdaFedSSL as training a deep neural network model over 100 clients. We assume a curious and trusted server that will not host data but only dealing with averaging over parameters or updates. At the beginning round $t = 0$, server initializes global model $\boldsymbol{\theta} g^t$ and setup connections to all users. Then, it randomly selects S number of users as partial clients who may be unqualified and rejected from training due to connection failure or unstable network before global aggregation. The number of users who finally participate in aggregation may not equal the number of users who are selected at the beginning if clients drop out. Server then broadcasts $\boldsymbol{\theta}_g^t$ to all $\mathcal{S}$ as local replicas $\mathcal{L}^t = \{\boldsymbol{\theta}_i^t, i \in (1, .., S)\}$. Many federated semi-supervised works [13, 14] only consider one type of user, namely, users are solely labeled or unlabeled. While in practice, end users may or may not have the motivation to label data. We assume three types of users in our framework: *labeled users*, *unlabeled users* and *semi-labeled users*. Local users start their training according to their data constitution. Suppose server selects S_l, S_u, S_s number of three types of users, respectively. *Labeled users* hold $B_i, i = (1, ..., S_l)$ number of labeled data $\mathcal{X}_i = \{(\mathbf{x}_b, \mathbf{p}_b) : b \in (1, ..., B_i), i \in (1, ..., S_l)\}$. Thus the global objective for labeled users is to minimize cross-entropy loss:

$$\ell_l^t = \frac{1}{S_l} \sum_{i=1}^{S_l} \frac{1}{B_i} \sum_{b=1}^{B_i} H(\mathbf{p}_b, p_{\boldsymbol{\theta}_i^t}(y|\alpha(\mathbf{x}_b))) \tag{2.12}$$

Notations are same with Section 2.2.1. *Semi-labeled users* hold $B_{i,l}$ and $B_{i,u}$ number of labeled samples $\mathcal{X}_i = \{(\mathbf{x}_b, \mathbf{p}_b) : b \in (1, ..., B_{i,l}), i \in (1, ..., S_s)\}$ and unlabeled samples $\mathcal{U}_i = \{(\mathbf{u}_b) : b \in (1, ..., B_{i,u}), i \in (1, ..., S_s)\}$, respectively. Referring to Equation 2.8, global objective for semi-labeled users is to minimize loss:

$$\ell_s^t = \frac{1}{S_s} \sum_{i=1}^{S_s} (\frac{1}{B_{i,l}} \sum_{b=1}^{B_{i,s}} H(\mathbf{p}_b, p_{\boldsymbol{\theta}_i^t}(y|\alpha(\mathbf{x}_b))) + \frac{1}{B_{i,u}} \sum_{b=1}^{B_{i,u}} \mathbb{1}(\max(\mathbf{q}_b) > \tau) H(\hat{\mathbf{q}}_b, p_{\boldsymbol{\theta}_i^t}(y|\mathcal{A}(\mathbf{u}_b)))) \tag{2.13}$$

Unlabeled users hold $B_i, i \in (1, ..., S_u)$ number of unlabeled samples $\mathcal{U}_i = \{(\mathbf{u}_b) : b \in (1, ..., B_i), i \in (1, ..., S_u)\}$, upon which we apply similar objective with semi-labeled users except its labeled loss. The training objective can be defined as:

$$\ell_u^t = \frac{1}{S_u} \sum_{i=1}^{S_u} \frac{1}{B_i} \sum_{b=1}^{B_i} \mathbb{1}(max(\mathbf{q}_b) > \tau) H(\hat{\mathbf{q}}_b, p_{\boldsymbol{\theta}_i^t}(y|\mathcal{A}(\mathbf{u}_b))) \tag{2.14}$$

Suppose at local iteration e, after loss computed, users backpropagate loss and calculate parameter update $\Delta_i^e, i \in (1, .., S)$, client gradient accumulator $(\mathbf{v}_i^e)^2, i \in (1, ..., S)$, and record the

number of newly generated pseudo-label $n^e_{i,u}, i \in (1, .., S)$. One may naturally ask why counting samples over batch basis. As the predicted probability of unlabeled samples varies over iteration, we mark unlabeled data as true when successfully generating pseudo-label. We sum up all true samples as the number of unlabeled data as the clent's weight. Let define the communication frequency as E, which means each E iterations of local update at each global round. Clients send Δ^E_i, $(\mathbf{v}^E_i)^2$ and $n_i = \sum_{i=1}^{E} n^e_{i,u} + n_{i,l}$ to server, where two aggregations are conducted:

$$\mathbf{v}^{t+1} = \frac{1}{S}\sum_{i=1}^{S}(\mathbf{v}^E_i)^2$$

$$\Delta^t = \frac{1}{S}\sum_{i=1}^{S} p_i \Delta^E_i$$

where $p_i = \frac{n_i}{\sum_{i=1}^{S} n_i}$ is the weight of i-th user in aggregation. The sum of $(\Delta^t)^2$ is denoted as $\mathbf{w}^t$. Given a global learning rate η_g, a global adaptive optimizer of Equation 2.5 updates global mode as $\boldsymbol{\theta}^{t+1}_g$. After the update, server broadcast $\boldsymbol{\theta}^{t+1}_g$ and $\mathbf{v}^{t+1}$ to users selected at next round. The whole process will repeat T number of rounds until reaching global optima. The algorithmic steps of our approach are shown in Algorithm 1.

2.4 Experiments

In this section, we conduct extensive experiments to evaluate our proposed AdaFedSSL on four benchmark datasets and compare related performance in different dimensions, including test accuracy and communication cost, with other benchmark methods. Additionally, we evaluate the effect of factors or hyperparameters (optimizer, Non-IIDness, weighting in aggregation, learning rate schedule, model selection).

Table 2.1: Test accuracy on MNIST, Fashion-MNIST, CIFAR-10 and SVHN over five federated semi-supervised learning methods. Boldface numbers indicate the best classification performance.

Methods	MNIST	Fashion-MNIST	CIFAR-10	SVHN
FedSSL	98.79	85.66	75.47	95.64
FedGrad	97.79	78.57	53.62	95.43
AdaAlter	97.28	80.56	65.29	93.92
FedAdaGrad	99.08	85.94	81.61	96.09
AdaFedSSL	**99.2**	**87.39**	**83.66**	**97.15**

Algorithm 1 AdaFedSSL: Adaptive Federated Semi-supervised Learning

Initialization: global model weight $\boldsymbol{\theta}_g$, server accumulator $\mathbf{w}_0 \geq \tau_g^2$, client accumulator $\mathbf{v}^0 \geq \tau_l^2$
for each round $t = 0, 2, \dots (T-1)$ **do**
  Select S number of training clients
  for each user $i \in 1, 2, \dots S$ in parallel **do**
    $\Delta_i^t, n_i^t, \mathbf{v}_i^t \leftarrow$ **Local_Training**$(\theta_g, \mathbf{v}^{t-1})$
  end for
  $n^t \leftarrow \sum_i^S n_i^t$
  $\Delta^t \leftarrow \sum_i^S \frac{n_i}{n} \Delta_i^t$
  $\boldsymbol{w}^t \leftarrow \boldsymbol{w}^{t-1} + (\Delta^t)^2$
  $\mathbf{v}^t \leftarrow \frac{1}{S} \sum_i^S \mathbf{v}_i^t$
  $\boldsymbol{\theta}_g \leftarrow \boldsymbol{\theta}_g + \eta_g \frac{\Delta^t}{\sqrt{\mathbf{w}_t + \tau_g^2}}$
end for

return $\boldsymbol{\theta}^g$

Local_Training$(\boldsymbol{\theta}_g, \mathbf{v}^{t-1})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_g, n \leftarrow 0, \mathbf{v} \leftarrow \mathbf{v}^{t-1}$
for $e \in 0, 1, \dots (E-1)$ **do**
  $\mathbf{g} \leftarrow \nabla H(\boldsymbol{\theta}, \mathbf{z}), \mathbf{z} \sim \mathcal{D}_i$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_l \frac{\mathbf{g}}{\sqrt{\mathbf{v} + \tau_l^2}}$
  $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{g} \circ \mathbf{g}$
end for
$n \leftarrow$ number of labeled examples

return $\boldsymbol{\theta} - \boldsymbol{\theta}_g, n, \mathbf{v}$

2.4.1 Datasets

Datasets used in our experiments are four image classification benchmark datasets: MNIST, Fashion-MNIST, CIFAR-10 and SVHN. All tasks have a 10-class data distribution for which we set the extent of IIDness to 10 levels as illustrated in Section 2.4.7. According to the number of images of each task, we split training samples into 100 balanced or unbalanced shards to represent 100 clients. The summary descriptions of these benchmark datasets are as follows:

- MNIST [90] consists of a training set of 60,000 samples and a test set of 10,000 samples of handwritten digits from 0 to 9, as shown in Figure 2.3a. Each example is a 28x28 gray-scale image associated with a label from 10 numbers. The number of training samples of each class is various and ranges from 5421 (class 5) to 6742 (class 1). The number of testing samples ranges from 892 (class 5) to 1135 (class 5).

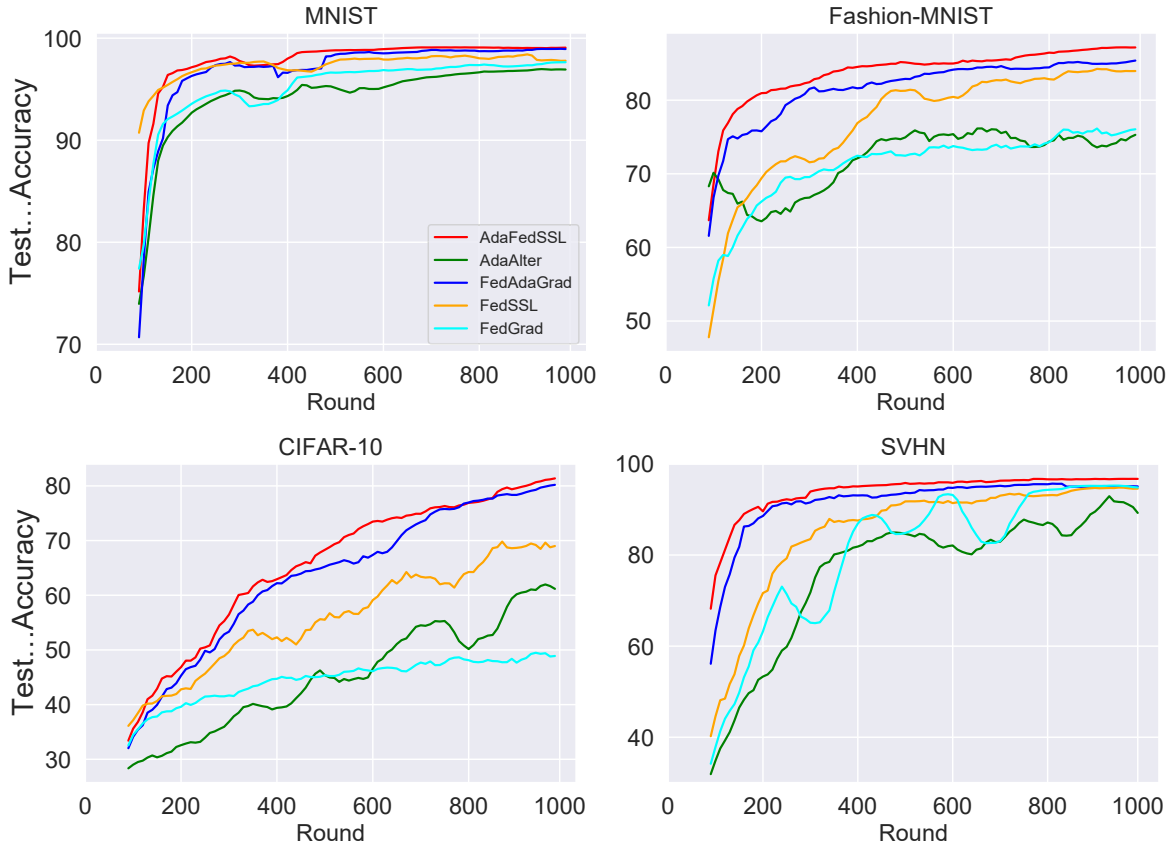

Figure 2.2: Comparison between AdaFedSSL and other baseline methods.

(a) MNIST (b) Fashion-MNIST (c) CIFAR-10 (d) SVHN

Figure 2.3: Sample images for four tasks.

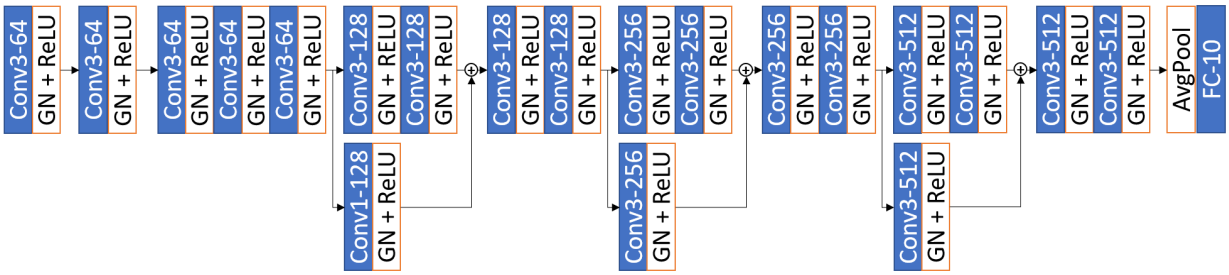

Figure 2.4: **Structure of ResNet-18 with Group Normalization.** Conv3-64 represents convolution layer with kernel of size 3 and output of 64-layer. GN stands for group normalization and ReLU is Rectified Linear Unit activation. FC-10 is a fully-connected layer with output of size 10.

- Fashion-MNIST [91] has 60,000 training and 10,000 testing MNIST-like images. Each example is a 28x28 grayscale image, associated with a label from 10 classes, as shown in Figure 2.3b. The number of training/testing images at each class is uniformly at 6,000/1,000.
- CIFAR-10 [92] consists of 50,000 training images with 5,000 images per class and 10,000 testing images with 1,000 images per class (Figure 2.3c). Each image has 3 channels (RGB) of 32x32 pixels. Each class has 5,000 training images and 1,000 testing images.
- SVHN [93] (training split) has 73,257 MNIST-like 3-channel 32-by-32 number images (Figure 2.3d) for training and 26,032 for testing. The number of training samples at each class is highly unbalanced, ranging from 4659 (class 9) to 13,861 (class 1). The number of testing images ranges from 1,595 (class 9) to 5099 (class 1).

2.4.2 Models

Models in our experiments have two versions, referred to as complex and simple models for each task. For CIFAR-10 and SVHN, we use ResNet-18 [94] with GN (grouping normalization) as the complex model. The model structure is shown in Figure 2.4. We train a 2-layer CNN as the simple model. For MNIST and Fashion-MNIST, we train a 4-layer multilayer-perceptron (MLP) as the complex model and a 2-layer CNN as the simplex model. The structure and number of parameters are detailed in Section 2.4.8. If not specified, all experiments take complex model at default. The two CNN models share the same structure with kernal size of 5 and each convolution layer is followed by a max-pooling and a rectified linear unit (ReLU) activation. We choose two models for each dataset in order to compare the effect of communication efficiency by reducing the model complexity. These models are not state-of-the-art on the corresponding datasets but are sufficient to show relative performance for the purposes of our investigation.

2.4.3 Implementation Details

We implement all experiments on a cluster of 2 machines with 8 GPUs in total. We use Pytorch Distributed Data-Parallel Training (DDP) to spawn parallel training. We use 2 CPUs (40 cores in total) of which one core represents one local computation. In case of 10 clients participating in training at each round, we trigger computation on 10 cores in parallel to simulate parallel training. We choose the first process to represent server that holds global accumulator and related global information. For the aggregation of model parameters and client accumulators, we directly use All-Reduce collective communication to update information locally. As each process cannot be allocated one exclusive GPU, we use 'gloo' as the communication backend. One GPU can host multiple processes and make each process possessing GPU alternatively. Apart from hardware settings, for federated training, our implementation has three distinctive features. First, the number of clients selected at each round is not constant. This simulates stragglers during training. Second, to account for the unbalanced quantity of training samples per client brought by pseudo-labeling, we weight the i-th model update Δ_i^t by Equation 2.11. We evaluate the effect of these two features by ablation study in Section 2.4.7 and Section 2.4.7. Thirdly, for unlabeled clients, if no pseudo-label is generated due to beneath of threshold $\hat{\tau}$, local training is skipped.

2.4.4 Experimental Settings

Except for ablation study, all settings are following the default in Table 2.2. We define the extent of IID among all clients as IIDness (definition is at Section 2.4.7) and set it as 3. For three types of clients, We set the number of unlabeled clients as 10, 10% of all clients for three types of clients. Other clients have a labeled client ratio, ranging from 0.1 to 1.0, representing the proportion of labeled samples at a client. For partial participation, the number of clients selected at each round is 10. For simulating stragglers, we randomly choose 7, 8, 9 or 10 clients to participate in aggregation. Global learning rate starts from 0.03 (Note a cosine annealing schedule with 5 warm-up rounds is applied). The degree of global adaptivity rate and local adaptivity rate both are 0.001. For settings of semi-supervised learning, we use two types of data augmentations (*i.e.*, weak and strong augmentations) referred to Section 2.3. Particularly, we abandon flip-and-shift augmentation from weak transformation on task MNIST and SVHN. This is because handwritten digits and street number digits are sensitive to angle; thus we keep the orientation of original images. Strong transformation is the same with RandAugment in work [95] that randomly selects several transformations from a given collection of transformations.

Table 2.2: Default value and description in basic setting.

Notation	Description	Value
T	The number of communication round	1000
K	The number of clients	100
K_u	The number of unlabeled clients	10
S	The number of participated client at each round	10
η_g	Global learning rate	0.03
τ_g	Global training adaptivity rate	0.001
B	Local batch size	64
E	Number of local iterations at each round	10
η_l	Learning rate of client	0.03
R	Number of classes at local client	3
r	Ratio of labeled data to all data at one client	0.1
τ_l	Local training adaptivity rate	0.001
μ	Relative size of unlabeled data to labeled data	5
$\hat{\tau}$	Threshold of pseudo-labeling	0.95
λ	Relative weight of unlabeled data to labeled data	1

2.4.5 Comparison with Other Methods

We compare AdaFedSSL with the other four methods, *FedSSL*, *FedAdaGrad*, *AdaAlter* and *FedGrad*. One important characteristic that distinguishes ours from other methods is the employment of an adaptive optimizer (*i.e.*, AdaGrad [89]) on both local and global sides. AdaGrad computes the $\ell 2$ norm of all previous gradients on a per-dimension basis. Learning rate is divided by this norm to make progress along each dimension even. It is very important to distributed deep neural network training since heterogeneous clients and partial participation easily cause overfit on certain dimensions during training. Based on optimizer type and location, we choose four baseline methods and provide a brief description below:

- *FedSSL* simply takes *FedAvg* as the global objective solver. Both server and client use SGD as optimizer with a constant $\eta_g = 1$ and a tunable η_l. η_l is a scalar hyperparameter on which techniques such as annealing schedule can be applied. Value of η_l is server-controlled and uniform, which is broadcast to selected clients at each round.

- *FedGrad* is a distributed version of AdaGrad. Like *FedSSL*, server uses SGD with constant $\eta_g = 1$. Clients apply AdaGrad, compute their own $\ell 2$ norm of previous gradients, and keeps it locally without sharing between each clients.

- *AdaAlter* utilizes AdaGrad on local sides and SGD on server side. Unlike *FedGrad*, server of *AdaAlter* averages $\ell 2$ norms of selected clients and updates norms to all clients with this

value at each round.

- *FedAdaGrad* [50] applies AdaGrad on server side and SGD on clients. Global learning rate is moderated through server accumulator while local learning rate is uniform among clients. Only model updates are communicated at each round.

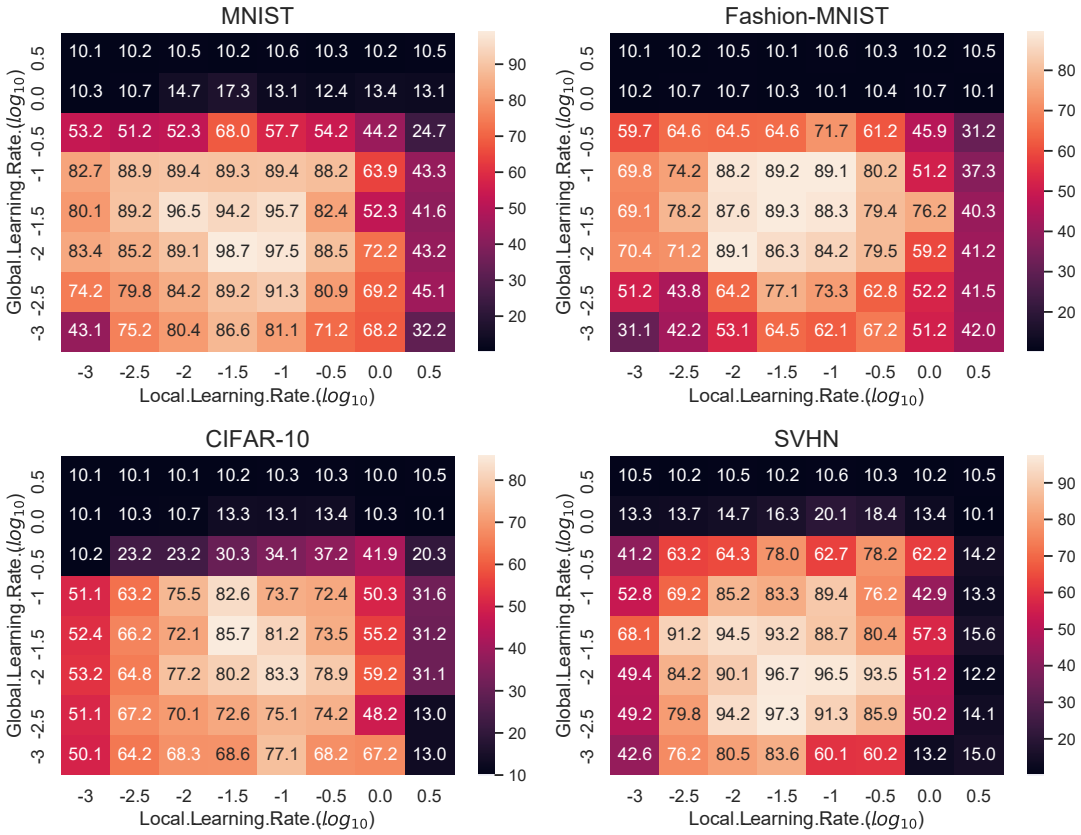

Figure 2.5: Test accuracy of various combinations of global learning rate and local learning rate on four tasks.

To measure model performance, as all tasks provide a test set, we quantify the performance as the accuracy over corresponding test sets. As shown in Table 2.1, our method outperforms others on all tasks. In specific, our method increases test accuracy of the best of others from (99.08, 85.39, 81.61, 97.15) to (99.2, 87.39, 83.66, 97.15) respectively on (MNIST, Fashion-MNIST, CIFAR-10, SVHN). Simply adopting an adaptive optimizer on clients aggravates training, as the values of *FedGrad* on all tasks are below *FedSSL*. As we illustrated above, *FedGrad* keeps client accumulator local and simply averages model update; thus, especially for partial participation, local optima could easily override global optima. Comparing *AdaAlter* with *FedGrad*, synchronization of client accumulator brings mitigation on this divergence. *AdaAlter* achieves comparable or lower results

on MNIST and SVHN while dramatically improving on Fashion-MNIST and CIFAR-10 compared with *FedGrad*. This is saying aggregating local gradient is of importance to certain tasks. We know the number of samples in MNIST and SVHN is relatively high and we get dense gradient of model parameters from later experiments. *AdaAlter* and *FedGrad* both perform worse than *FedSSL* on all datasets. This aligns with the notion that AdaGrad works poorly in dense settings. Another comparison between *AdaAlter* and *FedAdaGrad* gives that adaptive optimizer on server is more effective than that on clients. All accuracies improve from *AdaAlter* to *FedAdaGrad*. *FedAdaGrad* makes larger improvements on Fashion-MNIST and CIFAR-10 than the others (accuracy increments are (0.49, 6.28, 6.14, 0.46) on four tasks). Due to simple features in MNIST and a larger number of samples in SVHN, image features are well represented by local model. As a result, model updates are lightly different between each other and global accumulator takes less effect on moderating client drift. The accuracy rises significantly from *FedSSL* to *FedAdaGrad*, which indicates the effectiveness of global adaptive optimizer (server accumulator). From above, we conclude that sole local AdaGrad with or without synchronization can degrade model training while sole global AdaGrad benefit model training in dense setting. How about their combination? AdaFedSSL provides way better results than others. For a visualized comparison, We draw the learning curves of all methods by taking the average accuracy of a rolling window size of 10. As shown in Figure 2.2 our method shows a steady and the best learning curve.

2.4.6 Optimizers

In this part, we discuss two key elements that make AdaFedSSL successful: local and global AdaGrad. There are two important hyper-parameters in this optimizer, namely learning rate and adaptivity rate. We firstly grid search the best value of global learning rate and local learning rate. Then investigate the effect of applying learning rate decay on both or either of them. As for accumulator, same with learning rate, we grid search the best value of global and local adaptivity rates and study their mutual relation.

Local learning rate controls the extent of update at each local iteration, while global learning rate manages the extent of global update at each round. We conduct grid search on η_g and η_l and plot the accuracy with respect to each combination of them. As shown in Figure 2.5, we take grids of $\eta_l \in \{10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}\}$ and $\eta_g \in \{10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}\}$. Roughly looking at the the bright part of the images, we can observe a rectangular region, indicating many good pairs of learning rates are fit. This aligns with observation in work [50] that adaptive methods possess a certain amount of robustness when tuning η_g and η_l. Besides, the rectangular width is slightly larger than height, which

means more good values of η_l for each η_g and partly indicates client accumulator (width) adds more robustness into the learning process. Except Fashion-MNIST, all tasks have a rectangle at lower left, while FashionMNIST has it at upper left. This demonstrates a relatively higher global learning rate is more suitable for Fashion-MNIST task. As most optima present at or around the cross of $\eta_g = 10^{-1.5}$ and $\eta_l = 10^{-1.5}$, we take this pair as our default. Note that all experiments in this part are conducted with cosine annealing on η_g.

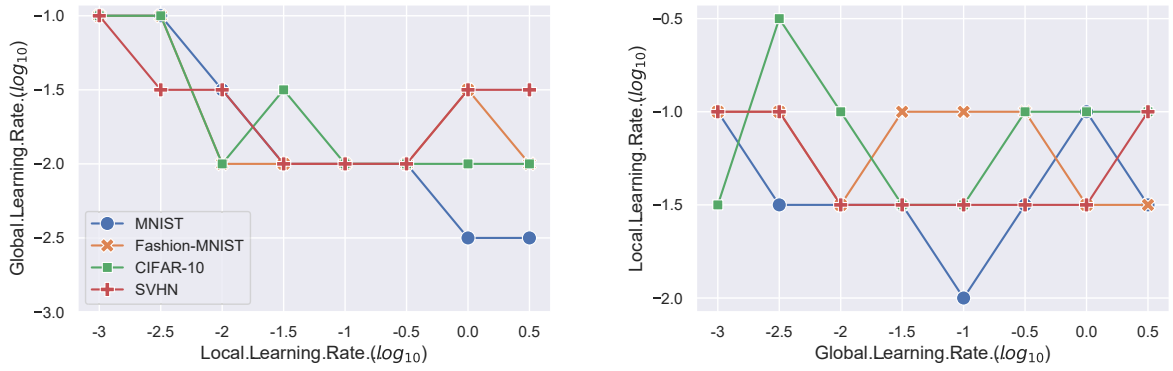

Figure 2.6: **Relationship between global learning rate and local learning rate on four tasks.** Right plot lists the best global value among gird for each local value. Left plot is the opposite.

One problem in tuning two hyperparameters (*i.e.*, η_g and η_l) is how to understand the **relationship** between them. We make two line plots that show the relation between optimal choices of these two parameters in Figure 2.6. For each task, we fix one learning rate and find the best corresponding the other one among the gird. From both plots, we cannot draw any obvious relation, though a coarsely inverse relation at the bottom. This observation testifies that both learning rates are stable in tuning because many good pairs of η_l or η_g can be selected when the value of the other fixes. In addition, a roughly inverse relationship is caught in the left plot, while the trend in the right plot is more constant. This indicates more suitable values of global learning rate when local learning rate varies, while relatively fewer values of local learning rate when global learning rate varies. This supports our observation in Figure 2.5 that the width of rectangular is larger than height (width represents global learning rate; height represents local learning rate). In general, learning rates in our approach are easy-tuning and demonstrate robustness in hyperparameter training.

Learning rate decay has been verified valuable in helping solver of gradient descent converge to critical point of its loss function. Traditional FL uses a generic learning rate schedule that is controlled by server. At each round, server computes the updated value and broadcasts it to each client. Due to this uniform value among clients, learning process can be inefficient if partial participation applies. For instance, learning rate becomes small at the end of training process if learning

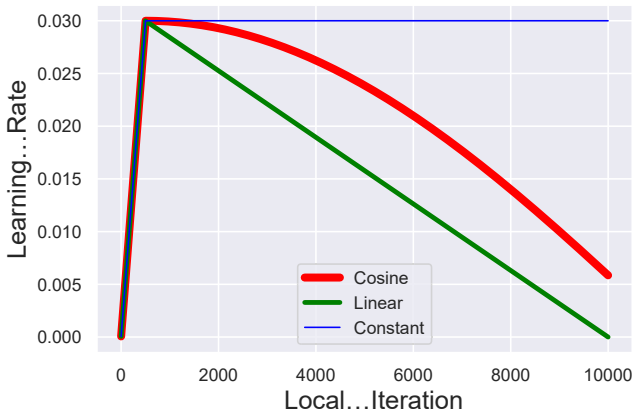

Figure 2.7: **Three types of learning rate schedules.** All schedules are time-based with 10000 total steps and start with a 500-step warm-up.

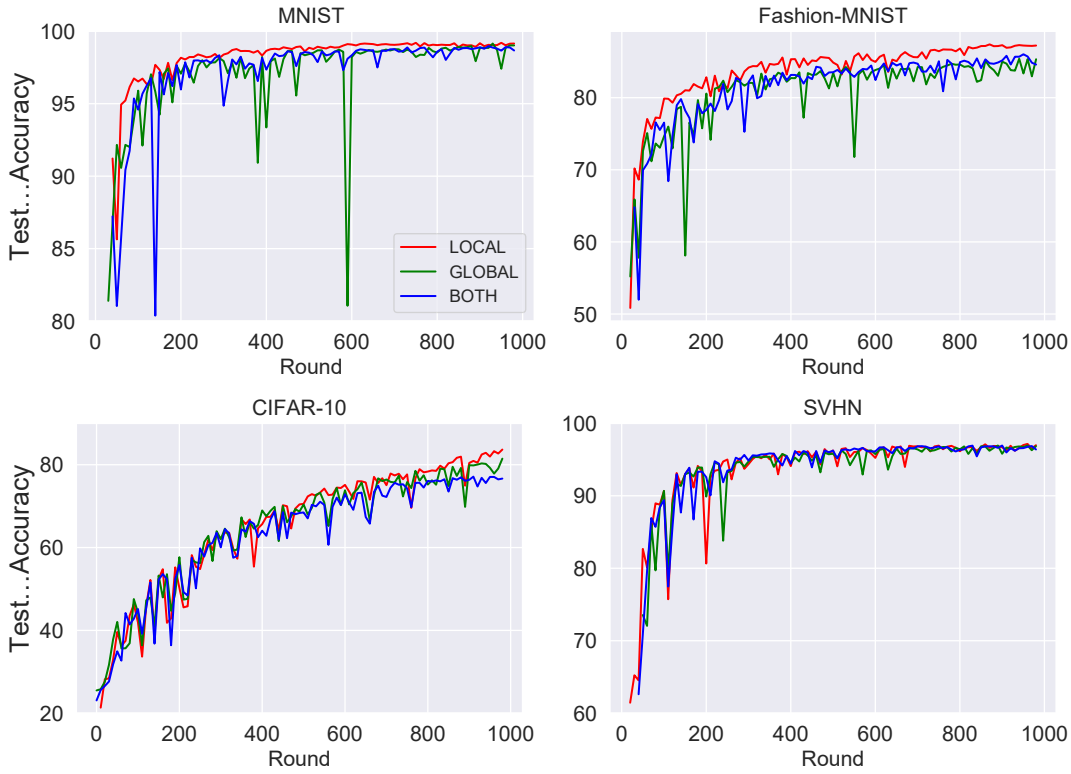

Figure 2.8: **Learning curves with cosine annealing schedule on different locations.** LOCAL and GLOBAL mean applying schedule on either local learning rate or server learning rate. BOTH stands for applying on both optimizers.

rate decay applies; thus newly-selected clients, who never participate in previous training, are unable to be well learnt. For adaptive optimizers, learning rate is automatically adjusted by dividing the norm. We consider applying different learning rate annealing schedules on AdaFedSSL and test the decay effect on adaptive optimizers. By default, a cosine annealing schedule is applied on η_g. We conjecture this technique brings effect on parameter tuning and model performance to some extent. We choose two common annealing approaches, cosine annealing and linear annealing, as shown in Figure 2.7, compared with a constant schedule. In AdaFedSSL, schedules can be applied on either or both η_l and η_g. Therefore we explore three scenarios, schedule on server side, schedule on local side and schedule on both sides. It is easy to understand how to apply an annealing schedule on server side. Analogous to centralized training, we multiply a scalar to η_g to update the value. As of local schedule, one approach is to compute update η_l on server and then broadcast the value among clients. Another approach is to set a personalized schedule that clients control. Due to different participation and random training sequences, it is reasonable to make schedules hold locally. However, in practice, this is unable to achieve as time-based schedules require a determined number of steps, which is unclear if we set schedules locally. In this case, We only experiment with the former approach.

Figure 2.8 shows the effect of applying cosines annealing schedule on either or both η_g and η_l. Except for SVHN, a clear improvement (blue line) is drawn when the schedule is employed on local side. Besides, the yellow line fluctuates significantly in the first two tasks. These observations partially explain that η_g is more robust in tuning than η_l as scheduling η_l makes a more obvious effect on training. For SVHN task, the number of training samples is huge, and the learning rate schedule is irrelevant to accuracy improvement. Applying decaying technique on both sides brings comparable results in MNIST, better result on Fashion-MNIST, worse result on CIFAR-10, compared with others. In this case, we discard applying schedule on both sides as it also introduces computation overhead.

Accumulator records the sum of the square of past gradients. To some extent, it represents the momentum of updates in certain dimensions. Device heterogeneity can be partly solved by adding regularization on local training objectives [96] and momentum on server [6]. We conjecture that introducing gradients accumulator on either global or local side may bring the same effect. Actually, the accumulator is a key element in AdaGrad. We instead compare adaptive methods with non-adaptive methods. A simple *FedSSL* uses SGD as both global and local optimizer. We separately compare *FedSSL* with the other two baseline methods: *FedAdaGrad* and *AdaAlter*. *FedAdaGrad* has a global accumulator recording the sum of the past Δ_t^2, which is the square of an averaged model update at a given round. *AdaAlter* has a local accumulator recording the sum

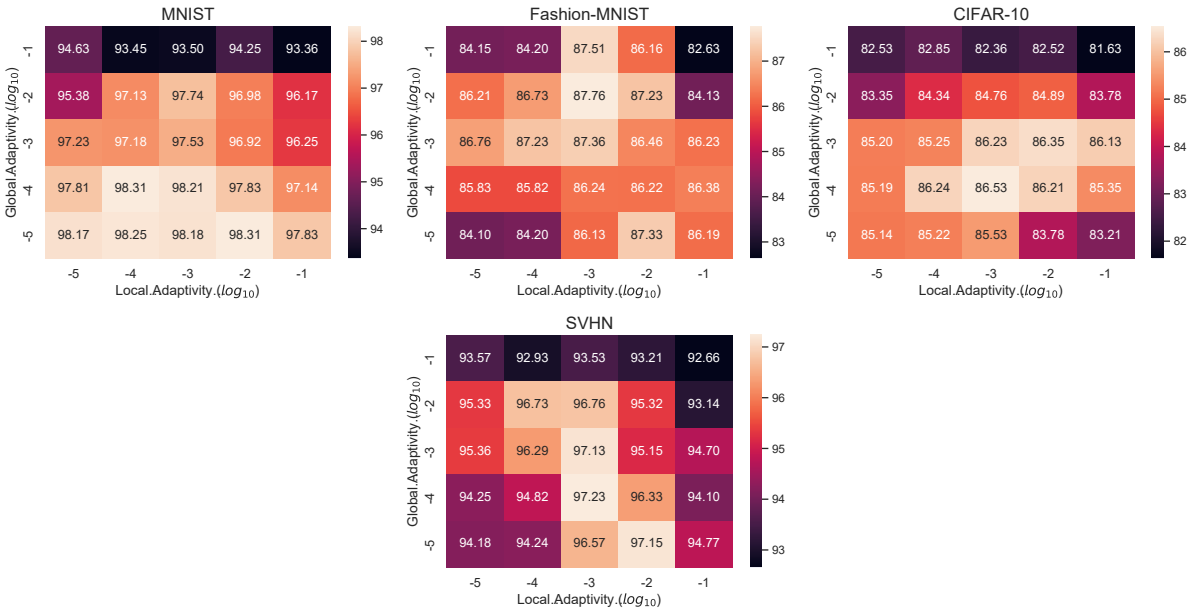

Figure 2.9: **Grid search results of global adaptivity rate and local adaptivity rate on four tasks.** The number is the accuracy of AdaFedSSL with various combination of local value and global value.

of past g^2, which is a square of gradients at a given iteration. Comparing red line and green line in Figure 2.2, all tasks are seen a noticeable improvement with global accumulator. It indeed triggers a global momentum that prevents global optima drifting. These observations align with the conclusion in [50] that global adaptive optimizer plays an important role in global convergence. Dissimilar with global accumulator, local momentum brings a negative effect on model training. This is because AdaGrad has been demonstrated ineffective in dense settings. The main reason why AdaFedSSL succeeds is the combination of these two accumulators. Local accumulator makes local training effective though it further fits local optima, while global accumulator fine-tunes divergence and accelerates convergence to global optima.

Another important hyperparameter in optimizer is adaptivity rates that determine the initialization and addition of accumulators. Server accumulator and client accumulator are initialized by τ_g^2 and τ_l^2, respectively. At each local iteration, τ_l^2 is the addition to the client accumulator. In synchronization, τ_g^2 is the addition to global accumulator. We wonder how these two parameters interplay and affect training just like we do of learning rates. Intuitively, they make effect directly on momentum as addition of accumulator. Figure 2.9 shows test accuracy as the function of τ_g and τ_l. We select both values from grid $\{10^{-5}, 10^{-}4, 10^{-3}, 10^{-2}, 10^{-1}\}$. As we can see, all tasks are non-tolerant to both high values of τ_g and τ_l. MNIST has better performance with low τ_g, while Fashion-MNIST is not robust with low τ_g. Both CIFAR-10 and SVHN show good values in the

middle of grid. Almost all tasks fit with $\tau_g = 10^{-3}, \tau_l = 10^{-}3$, and we take 10^{-3} as the default value for both τ. Figure 2.10 plots the relation between them. For each task, we fix one adaptivity and find the best corresponding the other one among the gird. Left plot shows the best τ_g when τ_l varies. There is an approximate direct relation between τ_l and the best τ_g. This also aligns with other observations that global AdaGrad offsets deficiency brought by local AdaGrad. No relation at right plot also indirectly testify the robustness of tuning τ_g.

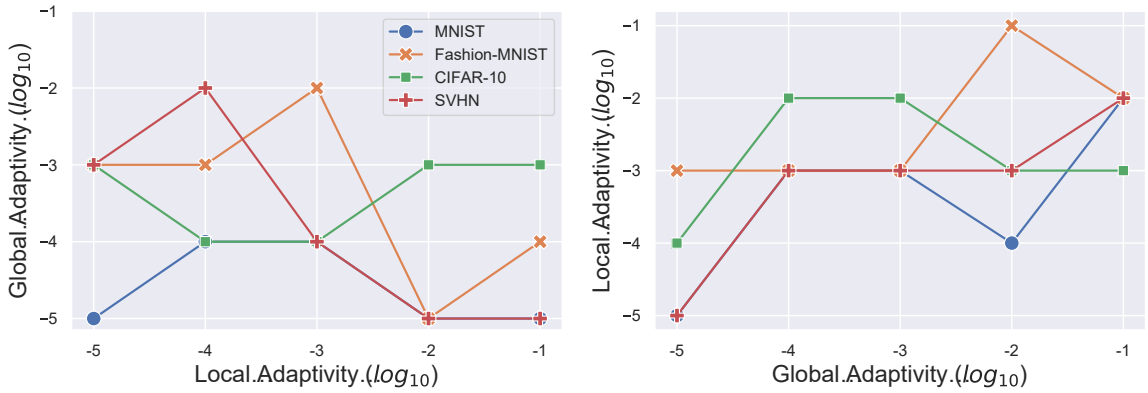

Figure 2.10: **Relationship between global and local adaptivity rate on four tasks.** Left plot lists the best τ_g in gird for each τ_l. Right plot is the opposite.

2.4.7 Statistical Concerns

We experiment with AdaFedSSL within various statistical settings, such as IID or Non-IID, weighting methods in model aggregation, stragglers during training and different amounts of labeled data. We define a variable to control the level of IID and acquire a direct relation between accuracy and this variable. For an unbalanced setting, we define the weight of aggregation as the actual number of training samples after pseudo-labeling. We also verified that AdaFedSSL is robust to stragglers. In the end, we evaluate the effect brought by lowering the amount of labeled data.

IIDness

Traditional federated learning assumes that distribution over classes on devices is IID because stochastic device update simulates stochastic gradient descent. However, it is unrealistic to make each device IID in Smart cities. *FedAvg* can work on its pathological Non-IID partition where each device only holds one to two classes while convergence speed is lowered. Work [3] specified distortion degree to 1-class and 2-class IID and suggested a shared global samples can improve accuracy dramatically. Jeong *et al.* [4] empowered server to train a conditional GAN's generator and replenish local device as IID. This approach is far away from ours that we assume server

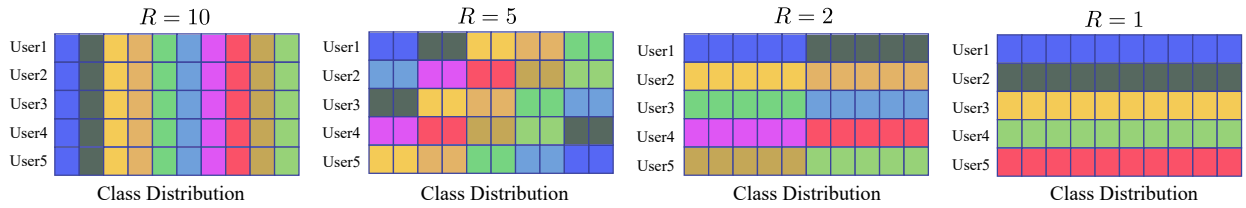

Figure 2.11: **Class distribution with different IID data.** Distribution among classes is represented with different colors. Note all settings have 100 users and the number of overall class is 10. R represents the number of classes in each user.

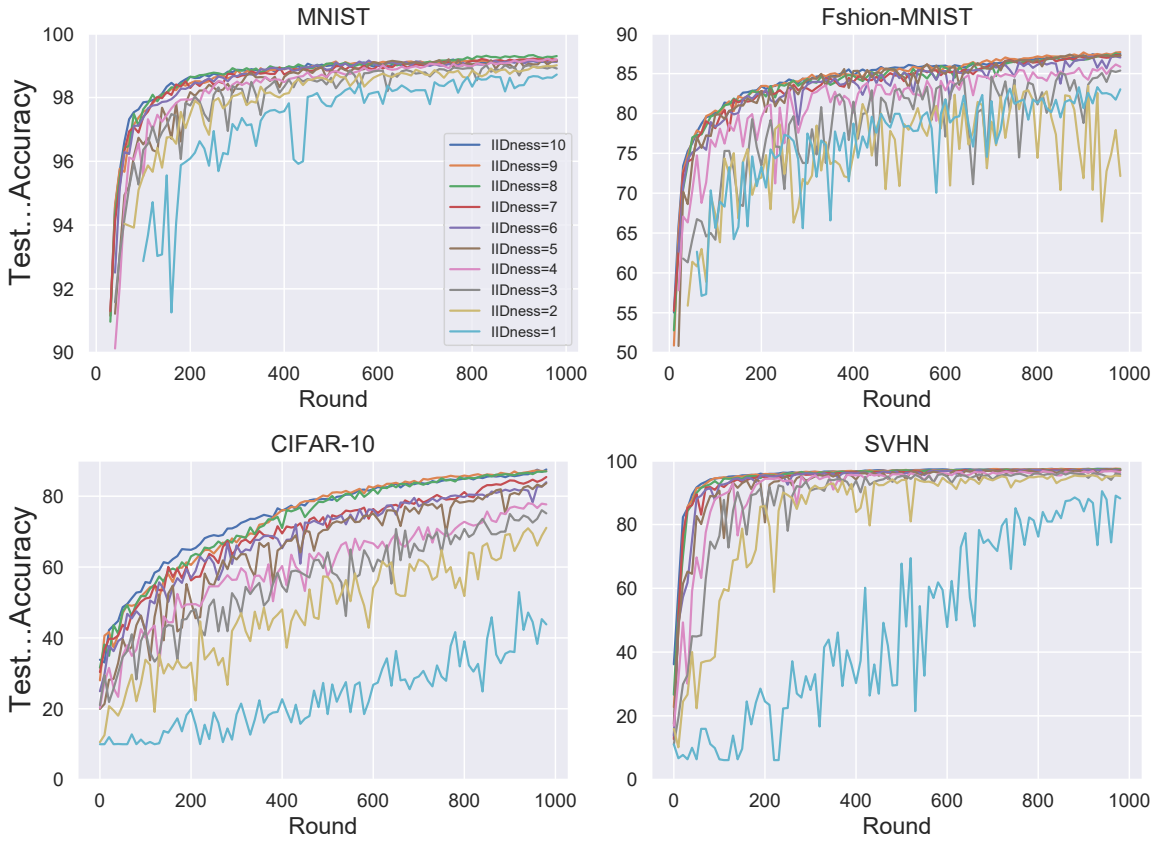

Figure 2.12: Learning curves of AdaFedSSL with various level of IIDness on four tasks.

only served as aggregator. Hsu *et al.* [6] took advantage of Dirichlet distribution which samples a predefined number of images from each class, whereas the number of samples must be balanced. Same with Dirichlet, Zhang *et al.* [13] assumes the same size of data at different devices that are not compatible with SVHN. We imitate work [3] to quantify the degree of distortion as the number of classes at a client. We use the term "IIDness", denoted as R, to show the number of classes at a client. As all tasks have 10 classes, R ranges from 1 to 10 to express IIDness from lowest to highest. In specific, $R = 1$ indicates each client only holds one class and is the most skewed distribution. Another extreme situation is $R = 10$ that each client holds all 10 classes which are referred to IID. Note that four datasets are using the same partition technique while the number of samples on each device may be different, *i.e.*, balanced or unbalanced. For example, the number of samples of each class is different for MNIST and SVHN. Given an R, we partition the entire training dataset to $100 \times R$ shards and give each client R shards. Each shard at a client is from different classes. To achieve this, We first sort the data by class; calculate the total number of shards needed in each class by multiplying R and the number of total clients; evenly partition the data into shards. Then we assign R number of non-identical shards to each client. Specifically, CIFAR-10 and Fashion-MNIST have partitioned 500 and 600 images respectively per client; MNIST and SVHN are unevenly partitioned. In Figure 2.11, each block denotes one shard of training data with different colors in different classes. We plot the test accuracy of IIDness from 1 to 10 in Figure 2.4.7 and plot the change of best accuracy over IIDness at the left of Figure 2.16. A clear performance degradation can be seen on both convergence speed and the value of accuracy when IIDness decreases. A more significant reduction in test accuracy and convergence stability is observed on CIFAR-10 and SVHN.

Table 2.3: Test accuracy of four datasets over different averaging methods. Boldface numbers indicate the best classification performance.

Methods	MNIST	Fashion-MNIST	CIFAR-10	SVHN
even average	99.02	85.38	76.04	96.35
sample-weight average	99.08	85.94	83.61	95.09
pseudo-label average	**99.20**	**87.39**	**83.66**	**97.16**

Impact of p_i

To testify the efficacy of our defined weighting from Equation 2.11, we compare model performance based on three scenarios. The first one is called even average, which means each model update Δ_i^t is evenly weighted, *i.e.*, $\frac{1}{S}$. The second method is sample-weight method. Each client's weight is defined as the number of training samples of the client divided by the total number of

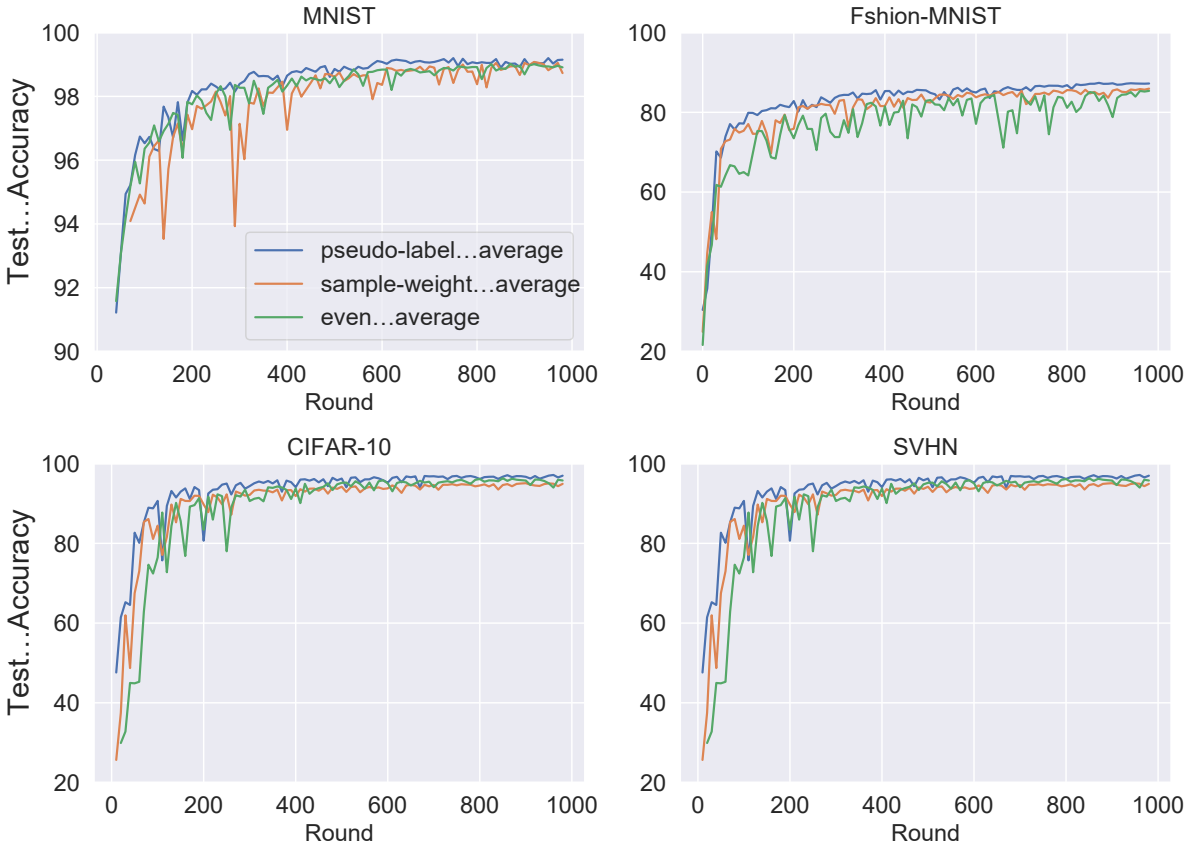

Figure 2.13: **Learning curves of AdaFedSSL with or without weigthed average of MNIST, Fashion-MNIST, CIFAR-10 and SVHN.** Pseudo-label average shows a stable and superior training performance among four tasks

training samples of all clients. In specific, under basic setting ($R = 3$), the number of samples on each client ranges from 573 to 629 of MNIST; from 498 to 501 of CIFAR-10; from 599 to 898 of SVHN. The number of Fashion-MNIST is uniform 600 among clients. The total number of training samples of all clients is articulated in Section 2.4.1. The third one is our defined pseudo-label average of Equation 2.11. The results are shown in Figure 2.13 and Table 2.3. Our method performs comparably or better than the other two methods on all tasks. Specifically, accuracy increases 2.01 and 7.62 compared with even average on Fashion-MNIST and CIFAR-10. Sample-weight average performs worse than even average on SVHN.

Impact of stragglers

Traditional FL frameworks assume that the connections of all training participants are always available. However, the connectivity may fail, or device energy drains, and thus device drops out from the FL system during training. This is associated with many problems such as unreliable device

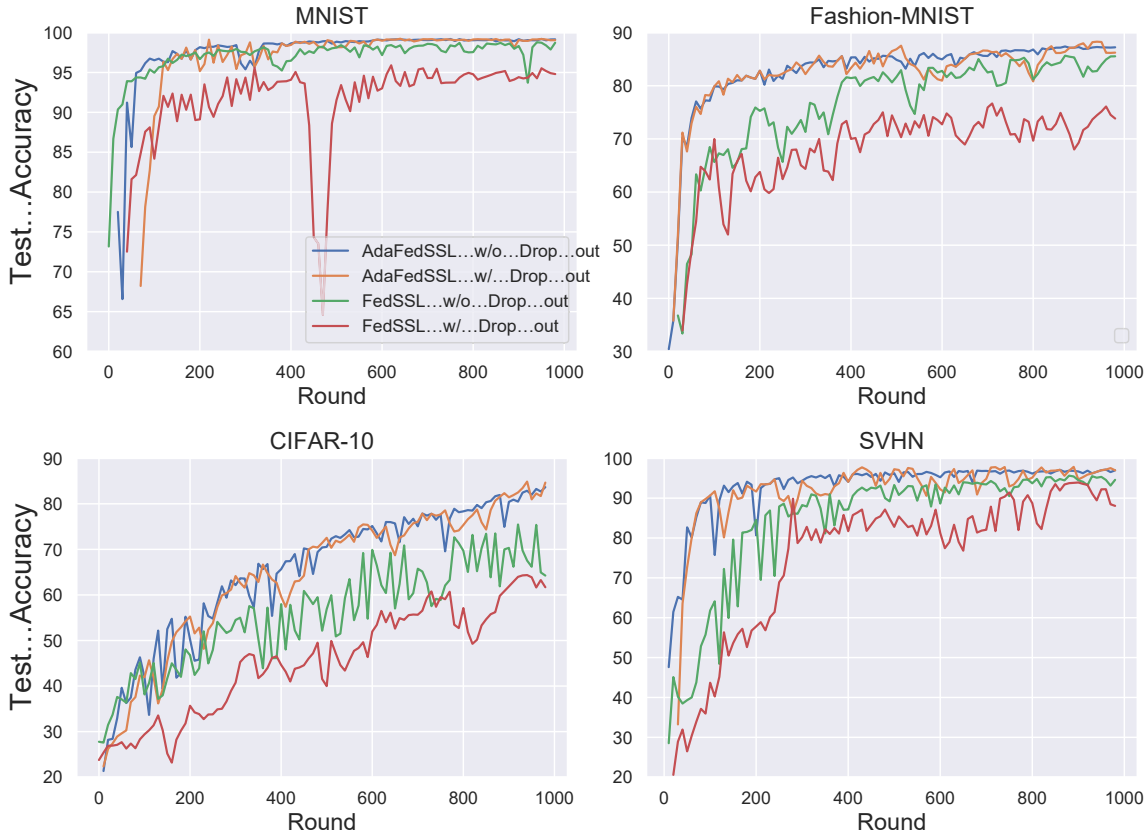

Figure 2.14: **Learning curve of AdaFedSSL and *FedSSL* w/ or w/o drop out for tasks of MNIST, Fashion-MNIST, CIFAR-10 and SVHN.** AdaFedSSL demonstrate stable accuracy when considering drop out in FL system.

connectivity and interrupted execution; orchestration of lock-step execution across devices with varying availability, limited device storage and compute resources [97]. To simulate stragglers, we anticipate the number of clients who participate in aggregation is smaller than the number of clients selected at the beginning of each training round. We conduct experiments based on four scenarios that show model performance of AdaFedSSL and *FedSSL* with and without stragglers. Note that *FedSSL* utilizes sample-weight method referred in Section 2.4.7. To be more specific, during each communication period, we randomly draw a number from $\{5, 6, 7, 8, 9, 10\}$ to represent the number of clients left. As such, for *AdaFedSSL*, on account of pseudo-label average, model performance would be close or better than situation without stragglers. We speculate a better performance as reduced client heterogeneity arisen from a reduced number of clients. As shown in Table 2.4, max accuracy increases from (99.20, 83.66) to (99.23, 84.88) in MNIST and CIFAR-10, respectively, after introducing stragglers clients. A slight drop is seen in Fashion-MNIST and SVHN but still higher than other methods. We plot the learning curve in Figure 2.14. The converge performance of drop-out for all tasks remains comparably or better than without drop-out. One notable obser-

vation is that clear performance degradation is seen in MNIST and Fashion-MNIST with regard to *FedSSL* when drop-out happens. We conjecture that a small number of clients could introduce or extrude heterogeneity from inter-device divergence. This also verifies that our algorithm is robust to device dropping out in the networks.

Table 2.4: Test accuracy of w/ or w/o stragglers on four tasks. Boldface numbers indicate the best classification performance.

Methods	Drop Out	MNIST	Fashion-MNIST	CIFAR-10	SVHN
AdaFedSSL	True	**99.23**	87.29	**84.88**	96.85
	False	99.20	**87.39**	83.66	**97.16**
FedSSL	True	95.89	76.67	64.36	93.91
	False	97.28	80.56	65.29	93.87

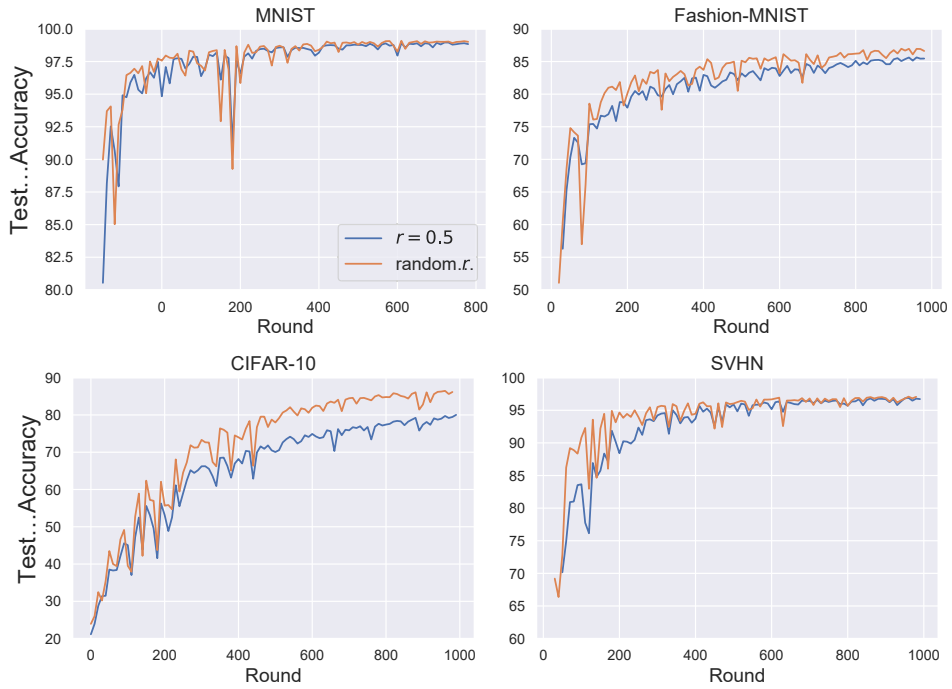

Figure 2.15: Learning curves with random amount of labeled data.

Amount of labeled data

We can quickly sort clients into three types, *i.e.*, totally labeled client, semi-labeled client and unlabeled client. However, how do we define the number of labeled samples at semi-labeled clients? Labeled and unlabeled clients are easy to get sorted while extra effort must be paid to quantify labeled clients. In reality, users may have various labeling practices. Some only label small parts

of data while some may label the whole. To consider this scenario, we assume the ratio of labeled samples to all samples at a client is $r \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. In our simulation, all clients randomly sample an r at initialization, and we make the corresponding amount of data labeled. For example, the number of labeled samples of $r = 0.1$ corresponds to 57 to 62 images in MNIST task, 60 images in Fashion-MNIST task, 49 to 50 images in CIFAR-10 task, 59 to 89 images in SVHN task per client. We take different random seeds and repeat the training 5 times. We plot the average training accuracy and compare it with the setting of $r = 0.5$. As shown in Figure 2.15, Fashion-MNIST and CIFAR-10 gain performance improvement with random labeling, while the other two almost keep intact. We conjecture that random r will potentially utilize more labeled data for all tasks while for MNIST and SVHN, training is already sufficient with lower r.

We conduct another experiment to verify a direct relationship between model performance and the number of samples. We all know a larger dataset makes centralized supervised training easier than training with scarce data. In this case, we make all clients semi-supervised clients and assign a unique r among clients to compare the effect of different values of r. The results are shown in the middle of Figure 2.16. We can see a clear increment of accuracy in task Fashion-MNIST and CIFAR-10 when labeled data scale-up, however nearly steady value in task MNIST and SVHN. This verifies our conjecture about random r making less effect to MNIST and SVHN. Other factors such as the number of unlabelled users K_u influence the overall amount of data. With increasing K_u, namely the overall amount of labeled samples decreasing, accuracy drops obviously. Detailed results are in Table 2.5 and right of Figure 2.16.

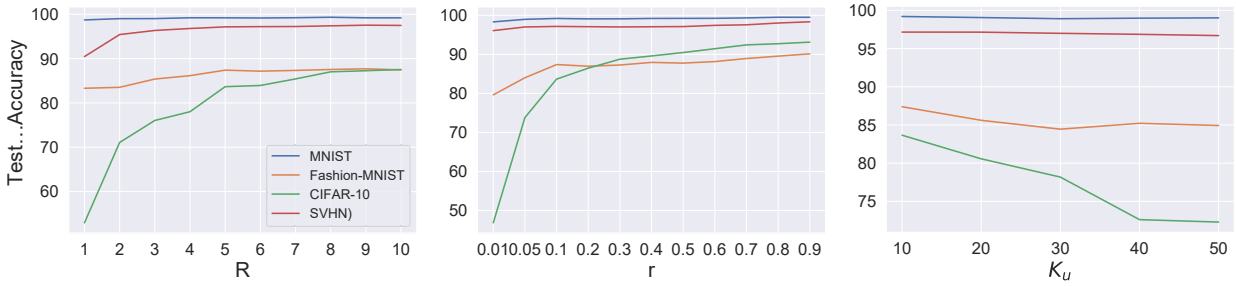

Figure 2.16: **Model performance with different statistical concerns.** Left shows relationship between model performance with IIDness. Middle plot draw the function of accuracy and the ratio of labeled data at one client. Right figure draws the performance decrements when the number of unlabeled clients increases.

2.4.8 Communication Efficiency

Communication Efficiency is one of the main challenges in FL. This concern is worsened by unreliable network conditions of participating devices and the asymmetry in internet connection speed

Table 2.5: Test accuracy related to the amount of labeled data in AdaFedSSL

K_u/r	CIFAR-10	MNIST	Fashion-MNIST	SVHN
10/0.1	83.66	99.2	87.39	**97.16**
20/0.1	80.75	99.06	85.61	97.13
30/0.1	78.18	98.91	84.45	96.99
40/0.1	72.61	98.98	85.22	96.86
40/0.1	72.29	99.02	84.93	96.70
10/0.01	46.85	98.32	79.63	96.08
10/0.05	73.82	98.97	84.01	97.02
10/0.2	86.46	99.09	86.96	97.09
10/0.3	88.76	99.1	87.27	97.02
10/0.4	89.59	99.2	**87.94**	97.08
10/0.5	**90.47**	**99.22**	87.76	97.11

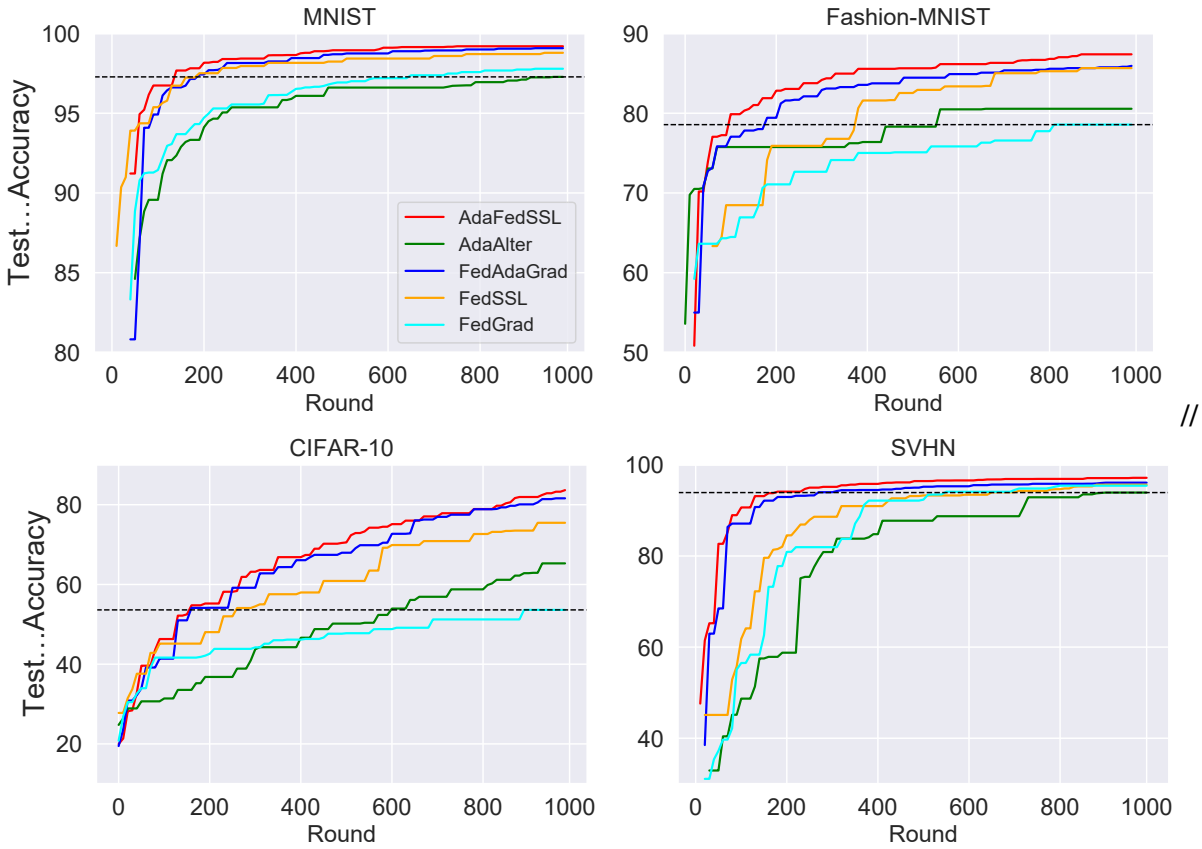

Figure 2.17: **Communication cost by using different methods on four tasks.** Dot line indicates target accuracy that is set by the smallest accuracy over five methods.

Table 2.6: $\hat{T}$ over differernt methods on four tasks. Boldface numbers indicate the lowest communication cost of each task.

Methods	MNIST	FashionMNIST	CIFAR10	SVHN
FedSSL ($\hat{T}\mathcal{W}$)	260	300	210	360
FedGrad ($\hat{T}\mathcal{W}$)	720	740	850	580
AdaAlter ($2\hat{T}\mathcal{W}$)	900	730	540	810
FedAdaGrad ($\hat{T}\mathcal{W}$)	320	80	**160**	**120**
AdaFedSSL ($2\hat{T}\mathcal{W}$)	**120**	**30**	120	80

in which upload speed is faster than download speed [7]. In our setting, each user trains a complex deep learning model that may contain millions of parameters, which contributes to a heavy communication burden. To solve this problem, we attempt to use different techniques, including decreasing communication frequency and reducing the size of model update, to analyze what methods are effective towards this concern. Mao *et al.* [86] concluded three approaches (*i.e.*, Edge Computation, Model Compression and Importance Based Updating) to cut communication costs. We only consider the first two methods in our case, as Importance Based updating involves comparing either gradients at different iterations or gradients at different locations, which is beyond our research scope. We first compare the communication cost of four baseline methods in the same configurations, namely having same model and communication frequency. Supposing our goal is to learn a model with parameters denoted by $\boldsymbol{\theta}$, at round t, model update at i-th client can be written as $\Delta_i^t = \boldsymbol{\theta}_i^t - \boldsymbol{\theta}_g^t$. Here $\boldsymbol{\theta}_i^t$ and $\boldsymbol{\theta}_g^t$ represent model parameters of i-th client and global model. Since *FedSSL* and *FedGrad* only communicate model parameters each round and the model update is of the same size as $\boldsymbol{\theta}$, we denote the communication cost of these two methods as $\hat{T}\boldsymbol{\theta}$ where $\hat{T}$ is the number of rounds reaching to the target accuracy. *AdaAlter* does share model update and shares client accumulator, which is of the same size with model parameters. We denote the cost of *AdaAlter* as $2\hat{T}\boldsymbol{\theta}$. *FedAdaGrad* computes global accumulator according to model updates, and thus only communicates $\hat{T}\boldsymbol{\theta}$. As such, *AdaFedSSL* will cost the same amount as *AdaAlter*. Even though the amount of parameters doubles, decreasing $\hat{T}$ would be another way to ease the burden. Target accuracy should be defined first for computing $\hat{T}$. We set the lowest test accuracy of experiments in a comparison group as the target. Then we make the learning curve monotonically improving by taking the best test accuracy over all prior rounds as the value of current round. We construct a continuous curve by using linear interpolation between discrete values and recording the round where the curve cross or tangents to target accuracy. As shown in Figure 2.17, the black dot line is our defined target accuracy. The round that achieves target is listed in Table 2.6. Note that we evaluate model at every 10 rounds resulting in all records of multiple of 10. *AdaAlter* spends the most

communication cost to achieve the target as the double size of model weight and highest number of rounds on all tasks except CIFAR-10. Compared with *FedSSL*, *FedGrad* performs worse due to increasing rounds of communication while *FedAdaGrad* is better on all tasks except MNIST. Our method shares the lowest number of rounds and halves the number of *FedAdaGrad* on MNIST and FashionMNIST while not effective on CIFAR-10 and SVHN, compared with *FedAdaGrad*.

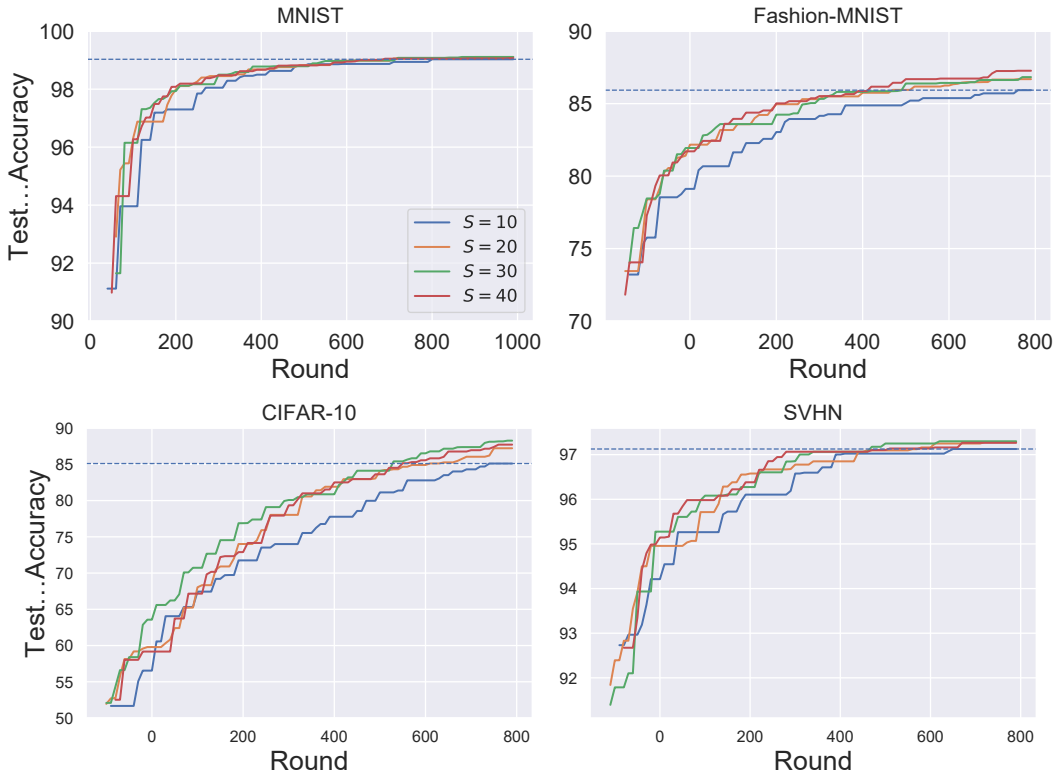

Figure 2.18: **Test accuracy over various client parallelisms on four tasks.** Dot line is the respective target accuracy indicated in Table 2.7. All experiments are with $F = 10$.

Decreasing communication frequency can be achieved by two ways: (i) parallelism increment and (ii) communication frequency decrements. Large parallelism means more edge devices participating in training per round, namely a larger S. We evaluate the model performance and the number of communication rounds to achieve target accuracy with various S on four tasks, as shown in Figure 2.18 and Table 2.7. Intuitively, with more participation, training would converge faster and more accurately. This is because increasing parallelism brings a larger batch size which improves the estimate of error gradients. As a result, for all tasks except MNIST, $S = 10$ has the lowest accuracy, and $S = 40$ generates the lowest communication cost. For MNIST and SVHN, a larger batch size only gains a small advantage (accuracy improvement smaller than 0.2) when increasing S from 10 to 20. Notably, the number of communication rounds remains steady in MNIST when S changes from 30 to 40. The accuracy even drops when S climbs from 30 to 40 in all tasks except

MNIST. The increment difference between two groups (MNIST/SVHN and CIFAR-10/Fashion-MNIST) aligns with our observations in Section 2.4.7. Better convergence occurs when $S = 30$ on CIFAR-10 and SVHN, $S = 20$ and $S = 40$ on MNIST and Fashion-MNIST, respectively. Simply increasing S is hard to shrink the number of communication rounds due to the large batch size, leading to a poor generation. Increased client participation only brings small advantage in the number of communication rounds while triggers more communication parallelism, which requires more computation locally. Based on this, we fix $S = 10$ as our basic setting. ($S = 20$ speeds up convergence rate efficiently while not halves the number of communication rounds compared with $S = 10$).

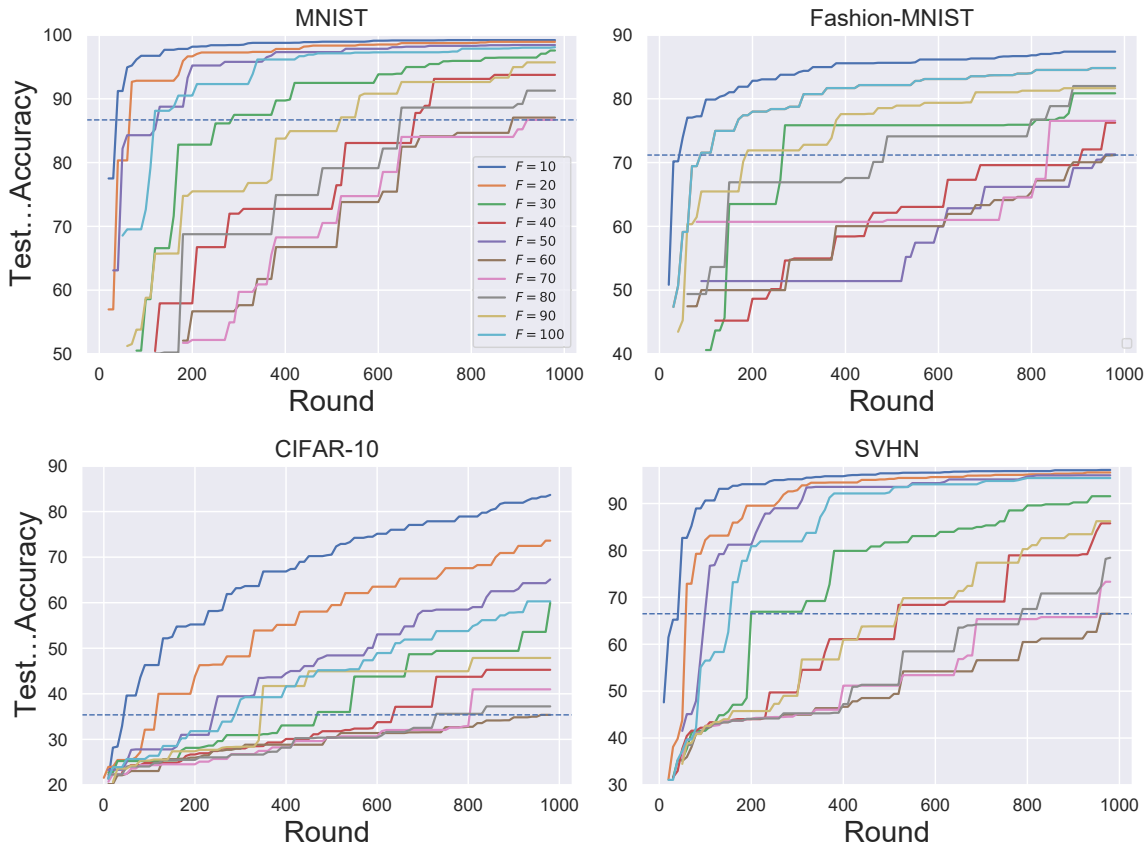

Figure 2.19: **Test accuracy over various communication frequencies on four tasks.** Dot line shows the respective target accuracy indicated in Table 2.7. All experiments are with $S = 10$.

Another way to lower communication costs is to use infrequent communication, *i.e.*, a larger F, which is the number of local computing iterations per round. Figure 2.19 gives the training curves with various F and demonstrates that $F = 10$ produces the best convergence result. Bottom part of Table 2.7 quantifies the communicating rounds and corresponding accuracy. All tasks gain the smallest number of communication rounds at $F = 100$, while at the cost of degraded model

performance. The number of communication rounds peaks at $F = 40$ and then directly goes down for MNIST while other tasks fluctuate when increasing F from 40 to 60. $F = 40$ is a proper setting for our experiments. There is no clear relation between the frequency of communication and the convergence speed. We can not draw any indirect relation between the amount to local computation and the number of communication costs. We conjecture the reasons behind this may relate to the sequence of training clients.

Table 2.7: Number of communication rounds to achieve target accuracy over different S and F. Test accuracies in parenthesis indicate model performance. Boldface numbers indicate the lowest communication cost of each task.

	CIFAR10	MNIST	FashionMNIST	SVHN
		$F = 10$		
$S = 10$	990 (85.11)	990 (99.03)	990 (85.93)	990 (97.13)
$S = 20$	840 (87.22)	900 (99.11)	810 (86.69)	920 (97.28)
$S = 30$	**750** (88.27)	890 (99.11)	810 (86.83)	**820** (97.29)
$S = 40$	790 (87.72)	**860** (99.07)	**720** (87.27)	930(97.26)
		$S = 10$		
$F = 10$	50 (83.90)	40 (99.20)	50 (87.39)	50 (97.17)
$F = 20$	60 (73.85)	35 (98.93)	45 (84.82)	30 (96.62)
$F = 30$	156 (59.98)	96 (97.81)	90 (82.13)	66 (91.56)
$F = 40$	160 (46.76)	170 (93.75)	227 (77.09)	130 (86.13)
$F = 50$	50 (65.11)	26 (98.43)	192 (83.11)	20 (96.06)
$F = 60$	160 (35.56)	148 (87.04)	163 (71.44)	160 (67.27)
$F = 70$	115 (40.98)	131 (88.23)	120 (71.40)	137 (74.83)
$F = 80$	91 (40.98)	81 (88.23)	61 (71.40)	98 (74.83)
$F = 90$	38 (47.87)	57 (95.71)	21 (82.00)	67 (86.24)
$F = 100$	**29** (60.29)	**12** (98.09)	**9** (81.78)	**16** (95.43)

Decreasing model update means less amount of data are sent to server at each communication round. Here we studied the trade-off between model performance and model complexity. For MNIST and Fashion-MNIST, we use a 4-hidden-layer multilayer-perceptron (MLP) with 512, 256, 256, 128 units and each layer uses a ReLU activation at the end. This is the complex model for these two tasks and the total number of parameters is 633226. We use a convolution neural network (CNN) with 2 5×5 convolutions layers (the first with 10 channels followed by a 2×2 max pooling and ReLu activation; the second with 20 channels followed by a 50% drop out, 2×2 max-pooling and ReLu activation), a fully connected layer with 50 units followed by a ReLu activation and a 50% drop out. This is referred as simple model and the total number of parameters is 21840. For CIFAR-10 and MNIST, the complex model is illustrated at Figure 2.4, which has 11173962 parameters. We use a CNN with two 5×5 convolutional layers (the first with 6 channels and the

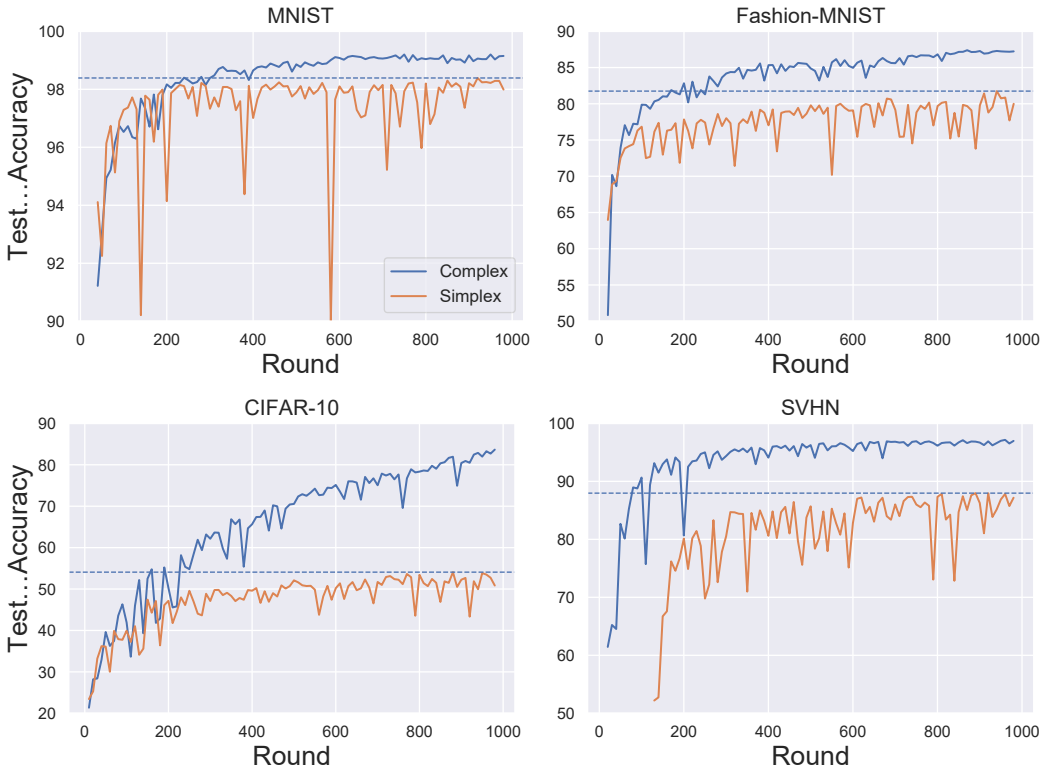

Figure 2.20: Learning curves over dynamic number of training devices.

second with 16 channels, each followed with a ReLu activation and a 2×2 max-pooling), two fully connected layers with 120100 units, respectively and each followed with a ReLu activation. The number of parameters is 64102. The number of parameters from complex model is 29 or 174 times larger than that of simplex model. We normally deem that a more complex model gives a better solution to the objective function. Here we study to what extent communication cost can be reduced by simplifying model structure while at the cost of deteriorating accuracy. We compare the convergence of complex and simple models in Figure 2.20. As all settings are unique except model type, the communication cost can be represented as the number of model parameters multiply the number of rounds of achieving target accuracy. The test accuracy declines form (83.66, 99.2, 83.79, 97.15) to (54.08, 98.39, 81.75, 87.97) of (CIFAR-10, MNIST, Fashion-MNIST,SVHN). The decrement (1% to 2%) is relatively small in MNIST and Fashion-MNIST compared with other tasks. Let us denote the target number of rounds required by complex model as $\hat{T}_c$, by simplex model as $\hat{T}_s$; the number of parameters in complex model is $|\boldsymbol{\theta}_c|$, in simplex model is $|\boldsymbol{\theta}_s|$. If we set the test accuracy of simplex model as target accuracy, we simply compare $\frac{\hat{T}_s}{\hat{T}_c}$ with $\frac{|\boldsymbol{\theta}_s|}{|\boldsymbol{\theta}_c|}$. If the former value is smaller, we can reduce communication costs by reducing model complexity. From our results, $\frac{\hat{T}_s}{\hat{T}_c}$ for all tasks are close but smaller than 10, which is far not enough to 29 and 174. To this end, if communication is the main issue and accuracy is second, largely

simplifying model may be the key.

CHAPTER 3

Human Activity Recognition

In this chapter, we apply adaptive federated semi-supervised learning to human activity prediction. First, we assume clients host an equal and generous amount of unlabeled data while the server has a small amount of labeled data. The training process is comprised of three main steps: client update, server aggregation, and classifier training. In the first step, clients collaboratively train autoencoders using adaptive optimizers in unsupervised learning and send model updates and accumulated gradients to the server. Second, the global autoencoder is updated via a gradient-based server optimizer to the average of clients' updates. In classifier training, labeled samples are encoded as latent representations by which a long-short term memory (LSTM) classifier is optimized by stochastic gradient descent in a supervised fashion. The server will broadcast the global autoencoder, averaged gradient accumulator to clients selected in the next round. Our framework is robust and resilient with partial participation and Non-IID data distribution. Our ablation studies show the effectiveness of the two adaptive optimizers. We also provide further insight into techniques such as data partition and partial updating, which play important roles in a successfull federated framework.

3.1 Introduction

Various application areas such as smart homes, human body monitoring, and smart cities use IoT technologies in modern life. A wide range of sensory data, including physiological data such as weight and blood pressure, ambient data such as occupancy, temperature, and brightness, is collected every moment by smart devices. These data are always in the form of time series data,

which is particularly helpful in training models for human activity recognition (HAR). Recently, HAR has become increasingly important for individuals who need in-home support or personal healthcare. By using IoT data, HAR can provide in-time and personalized assistance according to sensory data for improved qualiy of life. For instance, anomaly detection based on detected activity can trigger alerts when a person's health is deteriorating and thus early interventions can be adopted [98–100]. Medical specialists, for instance, can utilize long-term behavior change to determine dementia care [101].

One type of frameworks in HAR is utilizing the computational ability of edge devices distributed throughout IoT systems. Typically, these devices are responsible for collecting, storing, and processing data and have certain abilities to communicate with each other. This feature provides a possibility to collaboratively train or analyze human activity models under uniform deployment. To this end, federated learning is ideally suited with a centralized cloud server and can generate a general objective by aggregating the customized edge objectives in HAR. A traditional FL framework consists of two parts, namely global server and local clients, where clients separately train Deep Neural Networks (DNN), while the server aggregates local network parameters as the global model. This process allows customers to use local data to improve generality without privacy infringement. However, with the growing number and fast development of local devices, client heterogeneity and transmitting data between clients and server have become the main bottleneck that hinders IoT application development.

Most works of HAR take the premise that local clients only conduct supervised learning, where clients use labeled data to train a DNN model. However, this is always impractical due to the difficulty of acquiring labeled data.First, various sensors continuously generate time-series data; therefore, local computation is incapable of dealing with this huge amount of data. Second, time series data requires accurate preprocessing that is always time-consuming and needs input from experts. To this end, one of the main challenges of HAR in the federated learning setting is how to utilize unlabeled data properly.

An autoencoder [102] is one of the most commonly used techniques in unsupervised learning. It uses an encoder and a decoder to generalize compressed representation that can be used for further classifier training or feature extraction. Recent works (anomaly detection [103], Proactive Content Caching [104]) have shown autoencoder can be a useful local trainer to grasp local representation in an FL system. Autoencoder can be represented by a temporal convolutional network [105], Elman network [106], Long short-term memory [107] and so forth. The learned representations at local sides are then transferred to global server where further training is conducted. Results from autoencoder even produce close or better than that of supervised learning. Based on this, we

incorporate autoencoder in our current FL system to enable training with unlabeled data.

Considering realistic scenarios, we combine autoencoder with adaptive optimizers by conducting unsupervised learning at the edge while supervised learning at server. Besides the above-mentioned issue, we show our method is easy-tuned through our experiments even though the adaptive optimizer introduces adjustable hyper-parameters compared with non-adaptive methods. We also test our method on non-independent and identically distributed data (non-IID), one of the main challenges in traditional FL. Our results show robustness with Non-IID or unbalanced data. Our framework consists of three learning processes. First, local clients learn latent representation by training autoencoders with local unlabeled data. Then global server collects all local autoencoders and learns global autoencoder by adaptively aggregating local model updates. After that, server learns a classifier by using the encoded representation from global autoencoder which transforms labeled data into latent representation. Labeled data account for a small portion of the entire training data and are usually from the public datasets, which prevent clients' privacy leakage. We examine our framework on different benchmark datasets and compare our method with counterparts that replace key elements in our framework. Based on these points, the main contributions in this chapter can be summarized as follows:

- We introduce an adaptive federated semi-supervised learning method that employs an adaptive optimizer on both local and server sides. Our method yields a better mean F1-score compared with other baseline methods.
- Our framework is robust and resilient with partial participation and Non-IID data distribution. Ou experiments show a small amount of interplay among key hyperparameters and effectiveness on training with time series data.
- Our ablation studies show effects brought by key elements of AdaFedSSL. We provide insight into techniques such as data partition and partial updating in federated learning, which play important roles in a successful federated framework.

The rest of this chapter is organized as follows. In the next section, we present the problem formulation and propose our AdaFedSSL framework for human activity recognition. In Section 3.3, we present experimental results to demonstrate the competitive performance of our approach on three standard benchmark HAR datasets.

3.2 Adaptive Federated Semi-supervised Learning for Activity Recognition

3.2.1 Preliminaries

Autoencoder is a neural network that learns to reproduce its input as its output. It is an unsupervised learning algorithm that learns features from unlabeled data using backpropagation via stochastic gradient descent, and has typically an input layer representing the original data, one hidden layer and an output layer. An autoencoder is comprised of an encoder and a decoder, as illustrated in Fig. [2] [1]

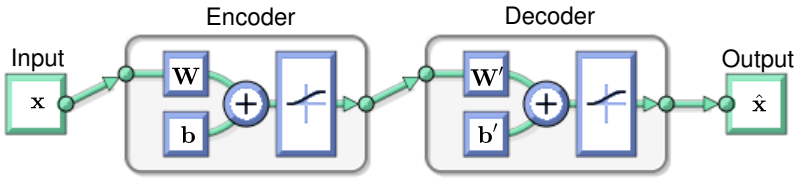

Figure 3.1: **Structure of a simple autoencoder with one hidden layer.** Input $\mathbf{x}$ is encoded as latent representation which has less number of dimensions after encoding process. Decoding process try to reconstructs $\mathbf{x}$ from latent representation and build $\hat{\mathbf{x}}$. Autoencoder learns key information required for reconstruction.

The encoder, denoted by $f_{\boldsymbol{\theta}}$, maps an input vector $\mathbf{x} \in \mathbb{R}^q$ to a hidden representation (referred to as code, activations or features) $\mathbf{a} \in \mathbb{R}^r$ via a deterministic mapping

$$\mathbf{a} = f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{3.1}$$

parameterized by $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$, where $\mathbf{W} \in \mathbb{R}^{r \times q}$ and $\mathbf{b} \in \mathbb{R}^q$ are the encoder weight matrix and bias vector, and σ is a nonlinear element-wise activation function such as the logistic sigmoid or hyperbolic tangent. The decoder, denoted by $g_{\boldsymbol{\theta}'}$, maps back the hidden representation $\mathbf{h}$ to a reconstruction $\hat{\mathbf{x}}$ of the original input $\mathbf{x}$ via a reverse mapping

$$\hat{\mathbf{x}} = g_{\boldsymbol{\theta}'}(\mathbf{a}) = \sigma(\mathbf{W}'\mathbf{a} + \mathbf{b}'), \tag{3.2}$$

parameterized by $\boldsymbol{\theta}' = \{\mathbf{W}', \mathbf{b}'\}$, where $\mathbf{W}' \in \mathbb{R}^{q \times r}$ and $\mathbf{b}' \in \mathbb{R}^q$ are the decoder weight matrix and bias vector, respectively. The encoding and decoding weight matrices $\mathbf{W}$ and $\mathbf{W}'$ are usually constrained to be of the form $\mathbf{W}' = \mathbf{W}^\intercal$, which are referred to as tied weights. Assuming the tied weights case for simplicity, the parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{b}'\}$ of the network are often optimized by minimizing the squared error

$$\ell_{auto} = \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \tag{3.3}$$

where N is the number of samples in the training set, $\mathbf{x}_i$ is the ith input sample and $\hat{\mathbf{x}}_i$ is its reconstruction.

Long short-term memory (LSTM) networks are a special type of recurrent neural networks (RNNs), capable of learning long-term dependencies between time steps of sequence data while being resilient to the vanishing gradient problem [108]. The key to an LSTM network is the cell state, which contains information learned from the previous time steps and has the ability to remove or add information using gates [109]. These gates control the flow of information to and from the memory. In addition to the hidden state, the architecture of an LSTM block is composed of a cell state, forget gate, memory cell, input gate and output gate, as illustrated in Figure 3.2. At each time step, the LSTM block takes as input the current input data vector $\mathbf{x}_t$ and both the hidden state (i.e. short-term memory) $\mathbf{h}_{t-1}$ and cell state (i.e. long-term memory) $\mathbf{c}_{t-1}$ from the previous cell. In order to decide which information to be retained or discarded at each time step before passing on the long-term and short-term information to the next cell, the LSTM block uses the forget, input and output gates, which are trainable functions with weights and biases. The forget gate decides which information from the long-term memory to forget, while the input gate can be regarded as a filter that selects what information can be kept and what information to be thrown out. The memory cell $\mathbf{g}_t$ is created by passing the current input and short-term memory into a $\tanh$ activation function, which is a shifted version of the sigmoid activation function. The new cell state $\mathbf{c}_t$ is obtained by adding two pointwise multiplication terms; the first term involves the input gate and memory cell, while the second one uses the forget gate and the previous cell state. The cell state $\mathbf{c}_t$ stores information about the input data across time steps. Finally, the hidden state $\mathbf{h}_t$ is obtained via pointwise multiplication of the output gate $\mathbf{o}_t$ and the new cell state through a $\tanh$ activation function. This hidden state (i.e. new short-term memory) is then passed on to the cell in the next time step.

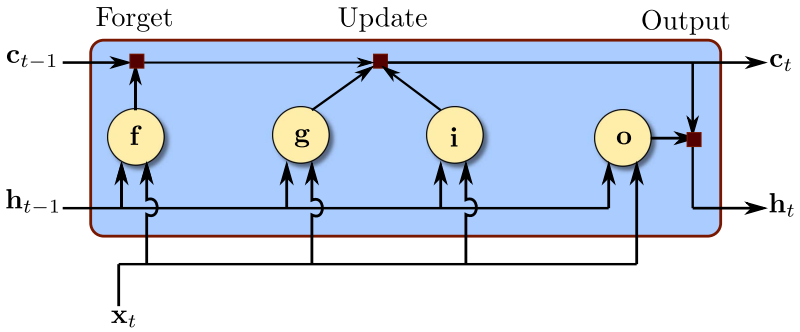

Figure 3.2: **Structure of a simple LSTM cell.** At time point t, a LSTM cell takes previous memory state $\mathbf{c}_{t-1}$, previous hidden state $\mathbf{h}_{t-1}$ and current data point $\mathbf{x}_t$ as inputs. The outputs are corresponding memory state $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$. σ denotes applying sigmoid activation and $tanh$ applies Tanh activation funtion.

Formally, given the input $\mathbf{x}_t$, current cell state $\mathbf{c}_{t-1}$ and hidden state $\mathbf{h}_{t-1}$ of the network, the

LSTM updates at time step t are given by

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{g}_t &= \tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{R}_g \mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{R}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{R}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{3.4}$$

where $\mathbf{f}_t$, $\mathbf{g}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$ and $\mathbf{h}_t$ are the forget gate, memory cell, input gate, output gate, cell state and hidden state, respectively; $\sigma(\cdot)$ denotes the sigmoid activation function; $\odot$ denotes the point-wise product; $\mathbf{W}_\bullet$ and $\mathbf{R}_\bullet$ are the learnable input and recurrent weight matrices; and $\mathbf{b}_\bullet$ are the learnable bias vectors.

In summary, the input gate controls what new information is added to cell state from current input, while the forget gate controls what information to throw away from memory. The output gate controls what information encoded in the cell state is sent to the network as input in the following time step. An LSTM network with multiple LSTM layers is referred to as a stacked or deep LSTM, with the output sequence of one LSTM layer forming the input sequence of the next. With the help of above three gates, LSTM networks enables to automatically choose relevant instead of all information to pass on. In our experiments, we treat a series of LSTM cell with the same length of encoded input, followed by a fully connected layer and a softmax layer, as classifier, which is trained in supervised learning.

Federated semi-supervised learning is applying semi-supervised learning in a federated way. Let us define the i-th client has labeled data $\mathcal{D}_i^L = \{(\mathbf{x}_b, p_b) : b \in (1, ..., n_l)\}$ where $\mathbf{x}_b$ is the b-th data sample and $p_b \in \{1, 2, ..., C\}$ is its corresponding label, where C is the number or classes and n_l is the number of labeled examples. Apart from labeled data, i-th client may also has unlabeled samples $\mathcal{D}_i^U = \{\mathbf{x}_b : b \in (1, ..., n_u)\}$, where n_u is quantity of samples. Generally the number of unlabeled data is large higher than the number of labeled data, *i.e.* $n_u >> n_l$, and in most cases of HAR, clients only have access to unlabeled data due to the fact of uneasy access to labeled data. In case that i-th client has both labeled and unlabeled data, local update is to minimize the loss:

$$H_i = H(\boldsymbol{\theta}_i, \mathcal{D}_i^L) + H(\boldsymbol{\theta}_i, \mathcal{D}_i^U) \tag{3.5}$$

where $H(\boldsymbol{\theta}_i, \mathcal{D}_i^L)$ and $H(\boldsymbol{\theta}_i, \mathcal{D}_i^U)$ are losses for labeled data and unlabeled data, respectively. Regarding to the aggregation scheme of *FedAvg*, the weight to each client is the sum of the number of both labeled and unlabeled samples, *i.e.* $p = n_l + n_u$. In case that server exclusively has labeled

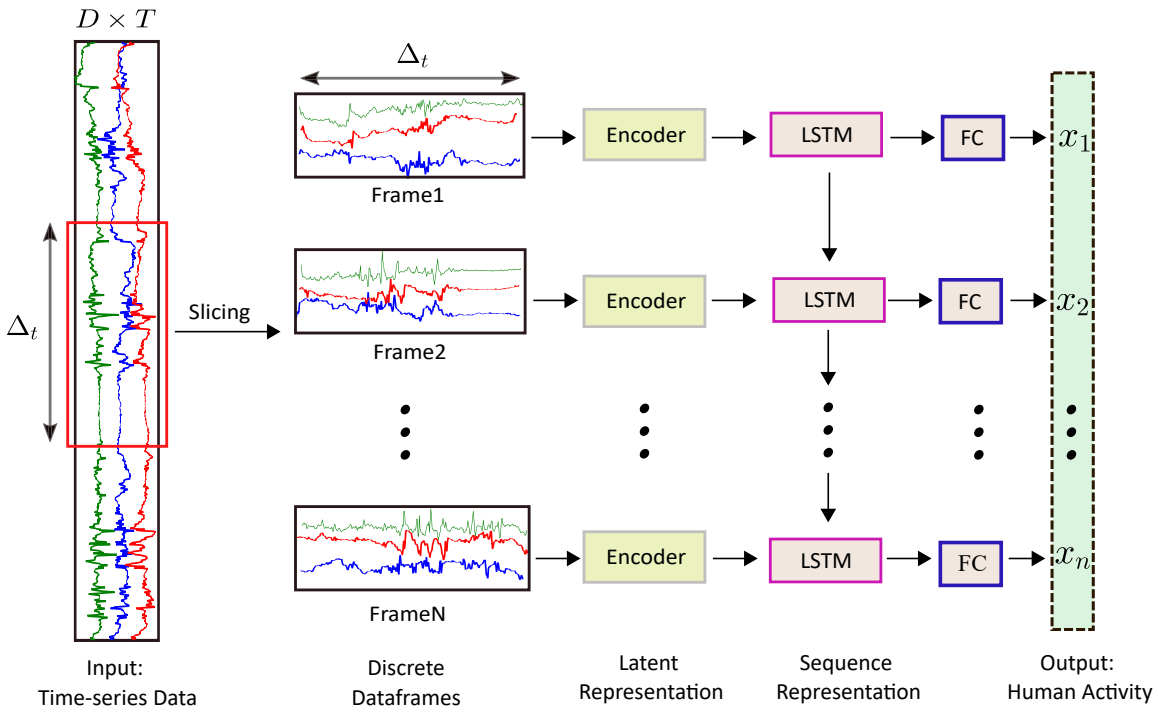

Figure 3.3: Architecture of Autoencoder and Long-short term memory (AE-LSTM) for human activity recognition. D is feature dimension of original data and Δ_t is the window size of slicing.

data $\mathcal{D}_s = \{(\mathbf{x}_b, p_b) : b \in (1, ..., n)\}$ where n is the number of labeled instances, and clients only store unlabeled data, we need calculate the loss twice. Supposing S number of clients is selected to participate in training, our global loss can be calculated by:

$$H = \sum_{i=1}^{S} H_i + H_g(\boldsymbol{\theta}_g, \mathcal{D}_s) \tag{3.6}$$

where $H_g(\boldsymbol{\theta}_g, \mathcal{D}_s)$ is supervised training loss on server where $\boldsymbol{\theta}_g$ is global model parameter. In practice, global server has a different model structure compared with local model if HAR framework separates whole process as a two-step, *i.e.* feature extraction and classification. They may share same parameters in holistic training where one model is trained twice sequentially on clients and server. In our framework, we this two-step version that an autoencoder is used in locally unsupervised training and a LSTM network is used for globally supervised training.

To completely form a HAR system, we combines the trained encoder, LSTM network and a fully connected layer end-to-end. First, original data is processed as latent features by encoder. Then the features are analyzed by LSTM network and classified by fullyconnected layer. The process is shown in Figure 3.3 and the specific prediction steps are:

- Encoder of a trained global autoencoder is utilized to extract high-level representation of processed data in higher dimension, *i.e.* $\{\mathbf{x}_i\}_{i=1}^{N}, \mathbf{x}_i \in \mathbb{R}^q$, where N is the number of frames

of time-series data and q is the number of feature dimensions. This latent representation, denoted as $\{\mathbf{a}_t\}_{t=1}^{N}, \mathbf{a}_t \in \mathbb{R}^r$ where r is the number of encoded feature dimensions.

- Latent representations pass LSTM series and become hidden state $\{\mathbf{h}_t\}_{t=1}^{N}, \mathbf{h}_t \in \mathbb{R}^h$ where h is the size of hidden dimensions, which is then fed into a fully connected layer followed by a softmax layers to generate distribution of class probability. We take the label with the largest probability as the prediction.

3.2.2 Proposed Framework

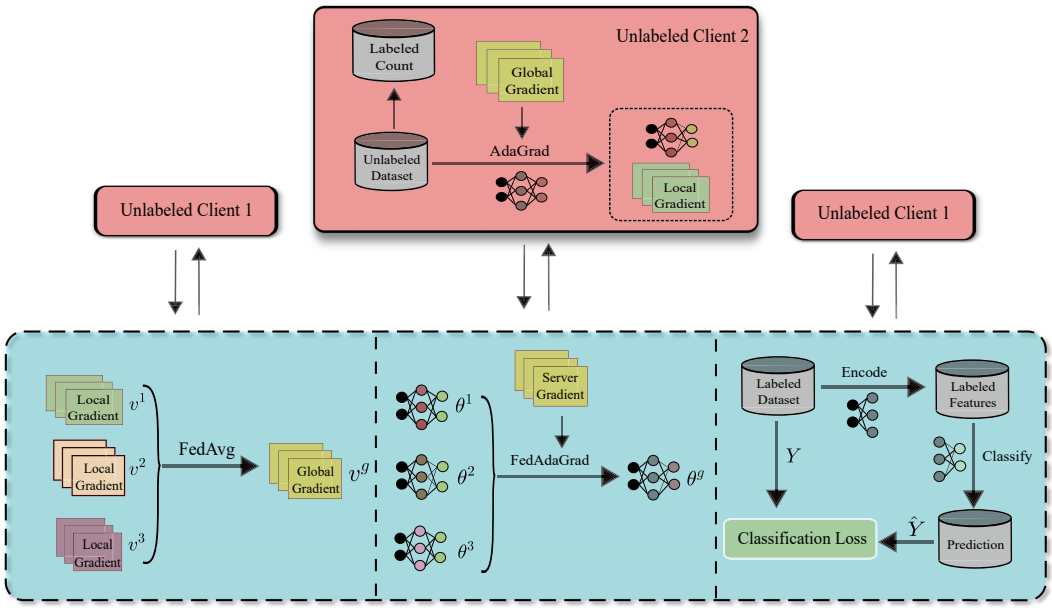

Figure 3.4: Proposed framework of AdaFedSSL for human activity recognition.

Overall training pipeline follows work [41] which conducts a two-stage training on clients and sever in sequence. We assume server hosts a small amount of data and is responsible for aggregating local autoencoders at each round. Two stages are shown in Figure 3.4. First, selected clients learn local representations by training local autoencoders with personalized unlabeled data. Autoencoders are sent back to server and aggregated as a global autoencoder. Second, server trains a classifier utilizing the encoded representation from global autoencoder in supervised learning, which repeats a predefined number of rounds and keeps the classifier generating the best performance. As server only requires a certain amount of labeled data which could be published or open data, client privacy is protected through this process. We use a simple autoencoder (with one hidden layer) to learn local representation and use an LSTM cell followed by a fully connected layer and a softmax layer as the classifier.

Initialization. At the beginning round $t = 0$, server initializes global autoencoder weight $\boldsymbol{\theta}_g^t$, classifier weight $\boldsymbol{\theta}_c^t$, server accumulator $\mathbf{w}^t \geq (\tau_g)^2$ and client accumulator $\mathbf{v}^t \geq (\tau_l)^2$, where τ_g and τ_l are global and local adaptivity rate respectively. Then server sets up connections to all 100 clients. As partial participation is our default training scheme, server randomly selects C fraction of overall clients, and $S = 100 \cdot C$ represents the number of selected clients. Server broadcasts $\boldsymbol{\theta}_g^t$ and $\mathbf{v}^t$ to all participated clients as local replicas denoted as $\mathcal{L}^t = \{\boldsymbol{\theta}_i^t\}_{i=1}^S, \mathcal{V}^t = \{\mathbf{v}_i^t\}_{i=1}^S$. As clients only conduct unsupervised learning, we term the participated clients as *unlabeled clients* who start local training with the same configurations such as the number of epochs and the learning rate value.

Autoencoder training. Supposing i-th client holds local dataset $\{\mathbf{x}_a\}_{a=1}^{N_i}, \mathbf{x}_a \in \mathbb{R}^d$ where each training sample is a sensory segment at a timestamp and d is the number of feature dimensions. At each training epoch, clients randomly initialize a batch size b that partitions the dataset of batches with size $n = \frac{N_i}{B}$. Thus, i-th client data set is represented as $\mathcal{D}_i = \{(\mathbf{x}'_a)\}_{a=1}^B$ where $\mathbf{x}'_a \in \mathbb{R}^{n \times d}$. As the batch size is randomized at the beginning of each epoch, the sequence length n is not constant. We aim to train an autoencoder in unsupervised learning by reconstructing the unlabeled dataset. According to Equation 3.3, we can write the local objective of i-th client as:

$$\ell_{auto,i}^t = \sum_{a=1}^{n} H(f_{enc}(\boldsymbol{\theta}_i^t, \mathbf{x}'_a), f_{dec}(\hat{\boldsymbol{\theta}}_i^t, \hat{\mathbf{x}}'_a)) \tag{3.7}$$

where H is loss function with input of original input vector and decoded vector. f_{enc} and f_{dec}, with corresponding parameters $\boldsymbol{\theta}_i^t$ and $\hat{\boldsymbol{\theta}}_i^t$, respectively apply a mapping of $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times p}$ and $\mathbb{R}^{n \times p} \to \mathbb{R}^{n \times d}$ where p is the dimension of latent features. Local encoders learn to extract key features of local datasets by updating parameters through back-propagating local loss. Let $\mathbf{g}_i^t = \nabla \ell_{auto,i}^t$ denotes the gradient computed by back-propagating loss upon $\boldsymbol{\theta}_i^t$. With AdaGrad, the optimization on i-th client is:

$$\boldsymbol{\theta}_i^{t+1} = \boldsymbol{\theta}_i^t - \eta_l \frac{\mathbf{g}_i^t}{\sqrt{(\mathbf{v}_i^t)^2 + \tau_l^2}} \tag{3.8}$$

where η_l is the local learning rate. Local optimizer is also updated by cumulating past gradients to update $(\mathbf{v}_i^t)^2$, where $(\mathbf{v}_i^t)^2 = \sum_{b=1}^B \mathbf{g}_i^b \circ \mathbf{g}_i^b$. After local training, participated clients calculate update $\Delta_i^t = \boldsymbol{\theta}_i^t - \boldsymbol{\theta}_g^t$ and send Δ_i^t, $(\mathbf{v}_i^t)^2$ to server, where two aggregations are conducted:

$$\mathbf{v}^t = \frac{1}{S} \sum_{i=1}^{S} (\mathbf{v}_i^t)^2$$

$$\Delta^t = \frac{1}{S} \sum_{i=1}^{S} \Delta_i^t$$

Given a global learning rate η_g and a global adaptivity rate τ_g, the global adaptive optimizer updates global autoencoder as:

$$\boldsymbol{\theta}_g^{t+1} = \boldsymbol{\theta}_g^t - \eta_g \frac{\Delta^t}{\sqrt{\mathbf{w}^t + \tau_g^2}}$$

where $\mathbf{w}^t$ is the accumulator of past $(\Delta^t)^2$.

Classifier training takes place after server aggregation. We choose an LSTM chain followed by a fully connected layer as the classifier. Supposing server has time-series data $\mathcal{D}_s = \{(\mathbf{x}_a, \mathbf{y}_a)\}_{a=1}^{N_s}$ where N_s is the length of time span, each instance on server is a pair of sensory signals at time a that $\mathbf{x}_a \in \mathbb{R}^d$ with d number of features and $\mathbf{y}_a$ is its corresponding one-hot label. Similar to autoencoder training, at each epoch, server randomly initializes a batch size B and batches dataset as a series of sequential data with length $n = \frac{N_s}{B}$. Labeled dataset can be rewritten as $\mathcal{D}_s = \{(\mathbf{x}'_a, \mathbf{y}'_a)\}_{a=1}^{B}$ where $\mathbf{x}'_a \in \mathbb{R}^{n \times d}$ and $\mathbf{y}'_a \in \mathbb{R}^n$. Batched data are encoded as latent representation at first which then is passed to LSTM classifier. Supposing at classifier training epoch t, the recurrent process is defined as:

$$\hat{\mathbf{y}}_i = f_{cls}(f_{eco}(\boldsymbol{\theta}_g^t, \mathbf{x}'_a), \boldsymbol{\theta}_c^t) \tag{3.9}$$

where $\boldsymbol{\theta}_c^t$ is the parameter of classifier. We use cross-entropy loss and SGD to update parameter which is formulated as:

$$\boldsymbol{\theta}_c^{t+1} = \boldsymbol{\theta}_c^t - \eta_c \nabla H(\hat{\mathbf{y}}_a, \mathbf{y}'_a) \tag{3.10}$$

where $H(\hat{\mathbf{y}}_a, \mathbf{y}'_a)$ is cross-entropy loss and η_c is classifier learning rate. Note that this loss is only used to update parameters of classifier, excluding autoencoder. After training classifier, server broadcasts $\boldsymbol{\theta}_g^{t+1}$ and $\mathbf{v}^{t+1}$ to users selected at next round. Detailed algorithm is shown in Algorithm 2.

3.3 Experiments

In this section, we conduct extensive experiments to evaluate the performance of our proposed AdaFedSSL on three HAR datasets and compare related performance in different dimensions, including mean F1-score and test accuracy, with other benchmark methods. Additionally, we evaluate the effect of factors or hyperparameters (fraction of partial participation, Non-IIDness, types of autoencoder).

Datasets. We conduct experiments on three benchmark tasks: OPP, DG, and PAMP2. These datasets have distinct features corresponding to different applications. Opportunity comprises recordings of normal daily activities such as drinking coffee and open/close drawer, that are short

Algorithm 2 AdaFedSSL for HAR

Initialization: global autoencoder weight $\boldsymbol{\theta}_g^0$, classifier weight $\boldsymbol{\theta}_c^0$, server accumulator $\mathbf{w}^0 \geq \tau_g^2$, client accumulator $\mathbf{v}^0 \geq \tau_l^2$, labeled dataset $\mathcal{D}_s = \{(\mathbf{x}_a, \mathbf{y}_a)\}_{a=1}^{N_s}$
for each round $t = 0, 1, \ldots T-1$ **do**
  Randomly selects S number of clients
  for each i-th user in parallel, $i = (1, 2, \ldots, S)$ **do**
    $\Delta_i^t, \mathbf{v}_i^t \leftarrow$ **Local_Training**$(\boldsymbol{\theta}_g^t, \mathbf{v}^t, \tau_l)$
  end for
  $\Delta^t \leftarrow \frac{1}{S}\sum_{i=1}^{S} \Delta_i^t$
  $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + (\Delta^t)^2$
  $\mathbf{v}^{t+1} \leftarrow \frac{1}{S}\sum_{i=1}^{S} \mathbf{v}_i^t$
  $\boldsymbol{\theta}_g^{t+1} \leftarrow \boldsymbol{\theta}_g^t + \eta_g \frac{\Delta^t}{\sqrt{\mathbf{w}^t + \tau_g^2}}$
  $\boldsymbol{\theta}_c^{t+1} \leftarrow$ **Classifier_Training**$(\boldsymbol{\theta}_g^t, \boldsymbol{\theta}_c^t, \mathcal{D}_s)$
end for

return $\boldsymbol{\theta}_g, \boldsymbol{\theta}_c$

Classifier_Training$(\boldsymbol{\theta}_g^t, \boldsymbol{\theta}_c^t, \mathcal{D}_s)$
$\boldsymbol{\theta}_a \leftarrow \boldsymbol{\theta}_g^t$, $\boldsymbol{\theta}_c \leftarrow \boldsymbol{\theta}_c^t$
for $e \in 0, 1, \ldots (E_c - 1)$ **do**
  Randomly initialize a batch size B
  $n \leftarrow \frac{N_s}{B}$
  $\mathcal{D}_s = \{(\mathbf{x}'_a, \mathbf{y}'_a)\}_{a=1}^{B}$
  for $(\mathbf{x}'_a, \mathbf{y}'_a) \in \mathcal{D}_s$ **do**
    $\hat{\mathbf{y}}_i \leftarrow f_{cls}(f_{eco}(\mathbf{x}'_a, \boldsymbol{\theta}_a), \theta_c)$
    $\boldsymbol{\theta}_c \leftarrow \eta_c \nabla H(\hat{\mathbf{y}}_a, \mathbf{y}'_a)$
  end for
end for

return θ_c

Local_Training$(\theta_g, \mathbf{v}^t, \tau_l)$
Local dataset $\mathcal{D} = \{\mathbf{x}_a\}_{a=1}^{N}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_g, \mathbf{v} \leftarrow \mathbf{v}^t$
for $e \in 0, 1, \ldots (E_l - 1)$ **do**
  Randomly initialize a batch size B
  $n \leftarrow \frac{N}{B}$
  $\mathcal{D} = \{(\mathbf{x}'_a)\}_{a=1}^{B}$
  for $\mathbf{x}'_a \in \mathcal{D}$ **do**
    $\hat{\mathbf{x}}'_a \leftarrow f_{eco}(\mathbf{x}'_a, \boldsymbol{\theta})$
    $\mathbf{g} \leftarrow \nabla H(\hat{\mathbf{x}}'_a, \mathbf{x}'_a)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_l \frac{\mathbf{g}}{\sqrt{\mathbf{v} + \tau_l^2}}$
    $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{g} \circ \mathbf{g}$
  end for
end for

return $\boldsymbol{\theta} - \boldsymbol{\theta}_g, \mathbf{v}$

and non-repetitive. Physical activity monitoring for aging people consists of physical activities such as bicep curls and raise leg to categorize the type of exercises (aerobic or strength). This type of signal is prolonged and repetitive in order to timely monitor and analyze user's physical condition. Daphnet Gait is used for automatically identifying gait freeze of Parkison patients. We describe details of three datasets and summarize them at Table 3.1:

- Opportunity (OPP) [110] consists of annotated recordings from a multitude of on-body sensors configured on four subjects while carrying out morning activities. The annotations comprise few types of locomotion along with a *Null* activity that makes classification more difficult. Data is collected while participants perform five activities of Daily Living (ADL) runs and a drill run of 20 repetitive activity sequences. Each instance refers to 113 real-valued signal measurements recorded at a frequency of 30Hz from 12 wearables on the body and is annotated with 18 mid-level gesture annotations (*i.e.*, open/close door). We only take data that have no packet loss in our training data, which include accelerometer recordings from the upper limbs and the back, and complete IMU data from both feet. Final dataset has 79 features (dimensions). We take runs 4 and 5 from subject 2 and 3 as our test set, while take run 2 from subject 1 as our validation set. The other data is used for training. For frame-by-frame analysis, We slice the data by a time window of 1 second and 50% overlap that thus create around 650k samples.

- Physical activity monitoring for aging people (PAMAP2) [111] consist of recordings from multiple wearables on 9 subjects that each of the subjects followed a protocol of 12 activities (lie, sit, stand, walk, run, cycle, Nordic walk, iron, vacuum clean, rope jump, ascend and descend stairs), and optionally performed a few other activities (watch TV, computer work, drive car, fold laundry, clean house, play soccer) as well. Over 10 hours of data were collected altogether from the 18 different activities. Data is collected from 3 inertial measurement units (Colibri wireless inertial measurement units) over the wrist on the dominant arm, on the chest and on the dominant side's ankle, respectively with sampling frequency of 100Hz and a heart rate monitor with sampling frequency around 9Hz. Final dataset has 52 features. We take runs 1 and 2 of subject 5 as our validation set and runs 1 and 2 of subject 6 as our test set. The other data is used for training. In order to have comparable temporal resolution with OPP dataset, data from accelerometer is downsampled to 33.3Hz. For slicing data, we use non-overlapping sliding windows of 5.12 seconds duration with one second shifting. From the segmented 3D-acceleration data, various signal features were calculated in both time and frequency domain (mean, variance, energy, etc.), and (normalized) mean and gradient are calculated on the heart rate data. The extracted features serve as input

for the next processing step, the classification. The resulting data consist of approx. 473k samples (14k frames) in training set.

- Daphnet Gait (DG) [112] consists of recordings from 10 patient with Parkinson's Disease (PD) while carrying out three kinds of tasks: straight line walking, walking with numerous turns, and finally a more realistic activity of daily living (ADL) task, where users went into different rooms while fetching coffee, opening doors, etc. Freezing is the temporary, involuntary inability to move that commonly happens in PD. The dataset is devised to benchmark automatic methods to recognize freezing incidents. Data is collected from several accelerations on ankle, leg and trunk, which comprise three classes (*i.e.*, no freezing, freezing and Null) with 9 features. We take run 1 from subject 9 as our validation set, runs 1 and 2 from subject 2 as our test set. The other data is used for training. Similarly, we downsampled the accelerometer data to 32Hz. For frame-by-frame analysis, we created sliding windows of 1 second duration and 50% overlap. The training set contains approx. 470k samples (30k frames).

Table 3.1: Three benchmark datasets for HAR

Dataset	Opp	PAMAP2	DG
Activity	Morning routine	Exercise	Parkinson
#features	79	52	9
#classes	18	12	3
#training	651K	473K	470K
#validation	119K	83K	81K

Models. The model consists of two parts, *i.e.*, an autoencoder and an LSTM classifier. We use the simplest fully-connected autoencoder with one hidden layer. The dimensionality of input and output accords with the number of features of each task. For input samples with n^f features, we take $r^f \in (0, 1)$ as compression ratio to control the level of encoding. The round value of $r^f \times n^f$ is the size of hidden features of autoencoder. We test with different r^f and different types of hidden layers, such as convolutional and LSTM autoencoder. Detailed comparison is shown in Section 3.3.5. Classifier on sever is a one-layer LSTM followed by a fully connected layer and a softmax activation. The input size accords with the output of encoder, and the hidden size is fixed at 1024.

Implementation Details. All experiments are carried out on a Linux server with one Intel Gold 6148 Skylake @ 2.4 GHz, 64 GB RAM, one NVidia V100SXM2 (16G memory) GPU card. For

comparing the best result of each baseline method, instead of using a uniform hyperparameter set, we tune hyperparameters separately on each method towards each task. Detailed tuning process is shown in Section 3.3.1. Based on the best setting, we repeat the experiments 10 times with a randomly split of data and take the average of metric as the final result. The partition and metric scheme are illustrated below. For training local autoencoder and server classifier, we take mean square error and cross-entropy loss as loss functions. Local and global autoencoders are optimized by AdaGrad while classifier is optimized by stochastic gradient descent. We implement all code based on Pytorch library.

Evaluation Metrics. We use mean F1-score as our basic metric to evaluate model performance. The highest possible value is 1.0, indicating perfect precision and recall, and the lowest possible value is 0 if either the precision or the recall is zero. Traditional F1-score or balanced F1-score is the harmonic mean of the precision and recall:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while Recall (also known as sensitivity) is the fraction of retrieved instances. Their corresponding definitions are:

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

where TP, FP, and FN denote true positives, false positives and false negatives, respectively. F1-score is typically interpreted in binary classification. For instance, in classifying whether a human is walking or not, TP is the number of correctly predicted human walking, while TN is the number of correctly predicted human not walking. A classifier that reduces FN (predict not walking while walking) and FP (predict walking while in fact not walking) indicates a better performance. In a general sense, precision measures the correct predictions among all positive predictions and reflects the level of trust in system. Recall is also called sensitivity and measures the true case in all positive cases. It is usually used in detecting disease detection. F1-score is a better measure to use if we need to strike a balance between precision and recall. Our tasks are multi-class classification problem, which requires extra effort to obtain overall F1-score. Traditional method for a balanced setting is to average F1-scores of classes. However, three datasets in our experiments are highly unbalanced (OPP: ranging from 4195 to 491072; PAMP2: ranging from 13790 to 60336; DG:

ranging from 40955 to 428485). We choose mean F1-score as the evaluation metric:

$$F_m = \frac{2}{C}\sum_{c=1}^{C}\frac{precision_c \cdot recall_c}{precision_c + recall_c}$$

where C is the number of classes.

Data Partitioning. Regarding federated learning, data partition is a key element for successfully simulating federated learning. As labeled data only resides on server, we adjust parameter r^l to control the ratio of labeled data distributed to server. For a training dataset with N samples, we adjust $r^l \in (0, 1)$ and take round value of $r^l \times N$ as the number of labeled samples on server. Though time-series data is processed into discrete data samples, the temporal relation is still expressed as the sequence of samples. To this end, we can not randomly pick $r^l \times N$ samples from overall samples. Instead, we first split training data into 100 chunks where each chunk retains the sequence of samples. Then we randomly select $100 \times r^l$ divisions and concatenate them as the final labeled dataset. This could largely avoid breaking the activity sequence while may cause some issues at division point. As for the rest data, we discard their labels and form unlabeled dataset for clients. Another partition regards distribution among clients. There are two strategies: IID and Non-IID. Supposing after generating labeled dataset, the number of unlabeled data is N^u. For a balanced setting, each client is supposed to have $n_k = \frac{N^u}{100}$ number of unlabeled data. For generating an IID local dataset, we firstly split unlabeled dataset into 100 chunks. Then for each chunk, we use a time window of length $\frac{n_k}{100}$ to contiguously generate 100 shards. Each client picks one shard from one chunk and combines 100 chunks into a local training dataset. Another strategy is Non-IID partition. For an IID distribution, we assume that the divided 100 chunks have same data distribution; thus local clients have the same distribution when they sample same amount of data from chunks. This is a systematic sampling that simulates local data as population distribution. For Non-IID partition, one simple way is randomly choosing partial chunks as the data source. For example, unlabeled data is still partitioned as 100 chunks, but for each client, we randomly selected $r \times 100$ chunks to generate $\frac{n_k}{r \times 100}$ number of time-series data concatenated as local dataset. We name $r \in (0, 1)$ as IIDness that controls the level of skewness. One extreme scenario that $r = 1$ denotes IID partition while $r = 0$ indicates the most severe Non-IID. We show the results of different r in Section 3.9.

3.3.1 Hyperparameter Tuning

Getting to the best performance includes tuning two learning rates η_g, η_l and two adaptivity rates τ_g, τ_l. First, we draw the model performance as a function of η_g and η_l. As shown in Figure 3.5,

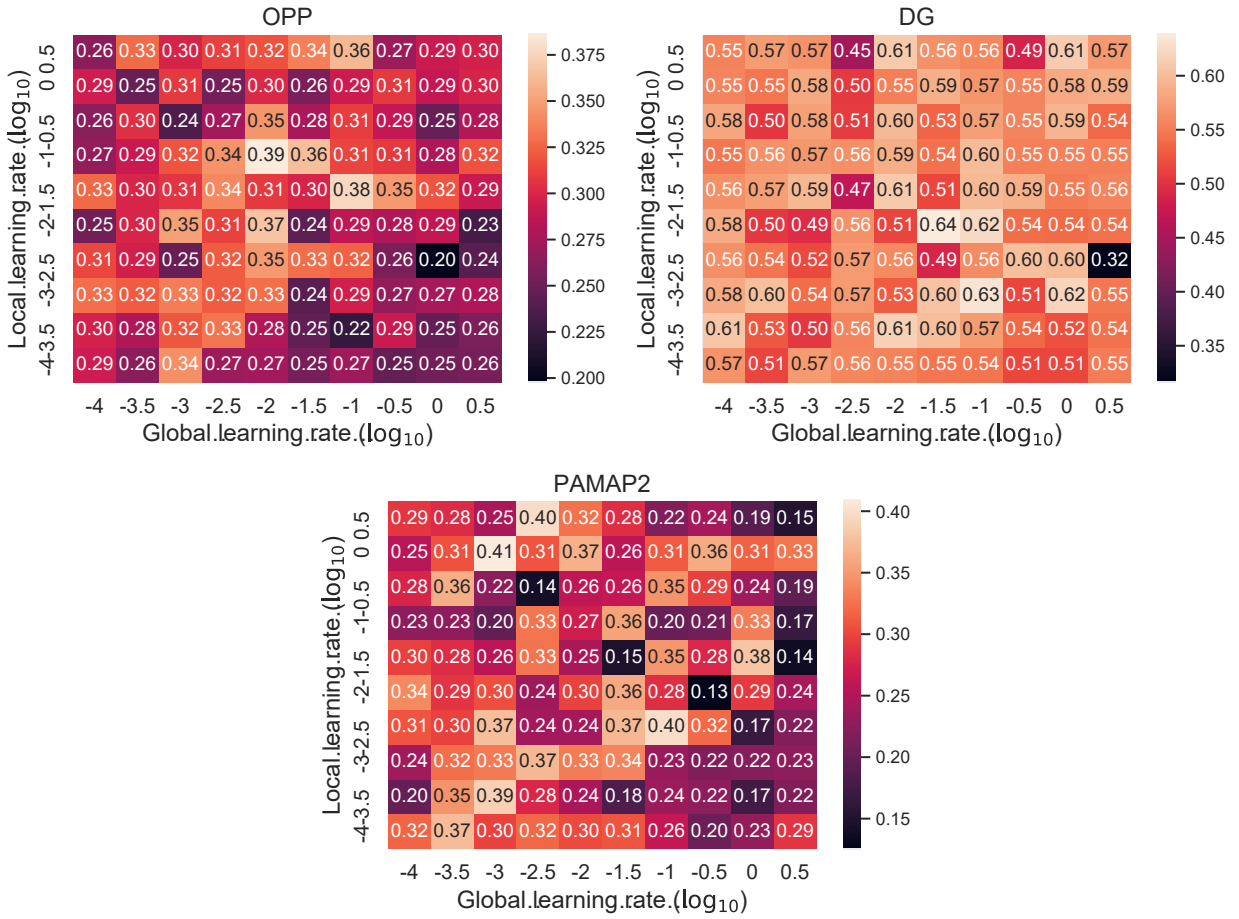

Figure 3.5: **Mean F1-score of AdaFedSSL with different combinations of global and local learning rate.** Global and local adaptivity rates are set as 10^{-5}.

we plot the distribution of model performance over different combinations of η_g and η_l. The most obvious observation is that most pairs on DG are good values, bringing a large area of light color. The best pairs locate at $\eta_g = 10^{-1.5}, \eta_l = 10^{-2}$ while pairs around it still result in considerable outcomes. There are fewer good choices on OPP that dark area is larger than that of DG. We can see the best situation of OPP is at $\eta_g = 10^{-2}, \eta_l = 10^{-1}$. Unlike OPP and DG, PAMAP2 reveals an unsystematic distribution of mean F1-score that tells an unsteady learning process. We take $\eta_g = 10^{-1}, \eta_l = 10^{-2.5}$ as the default value for this task.

After setting learning rates, we need to dig deeper to fine-tune the level of adaptivity. Similarly, we plot the mean F1-score as a function of global adaptivity τ_g and local adaptivity τ_l. The best value pair of OPP is $\tau_g = 10^{-5}, \tau_l = 10^{-3}$ and some other acceptable values appear when having small τ_l. The overall distribution of model performance seems not groups in a concentrated area. Better results are seen on DG, which shows more good pairs on the middle plot. The best value of the two parameters are 10^{-6} and 10^{-4}. PAMAP2 demonstrates a complicated scenario that only

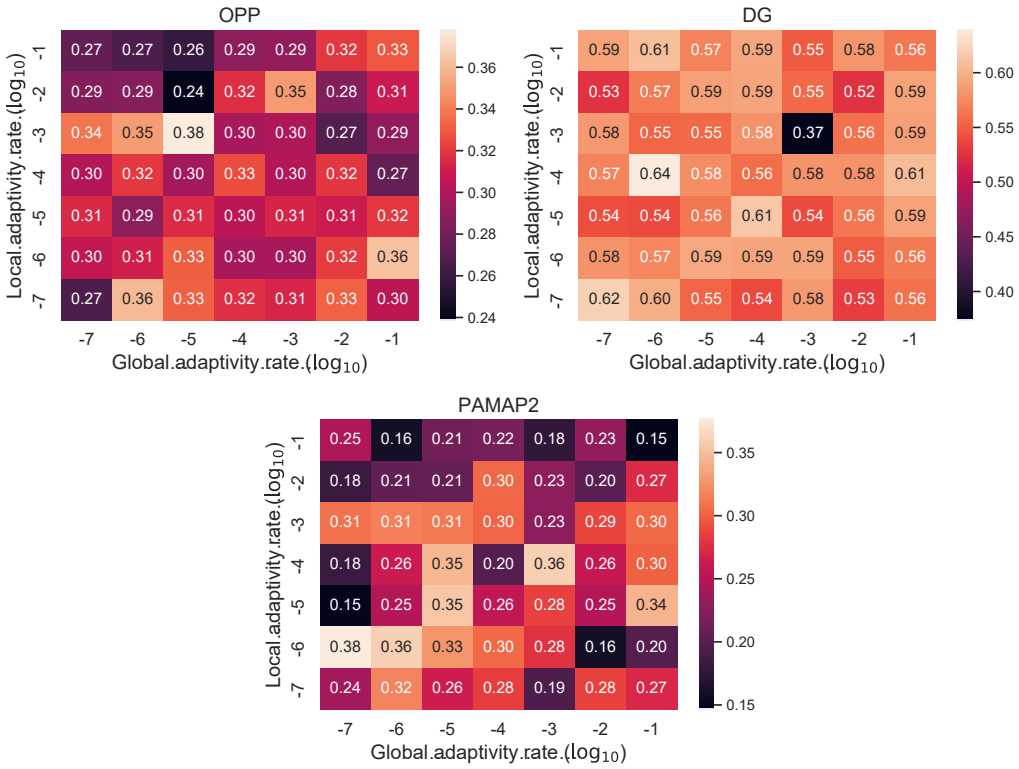

Figure 3.6: **Mean F1-score of AdaFedSSL with different combinations of global and local adaptivity rates.** Global and local learning rates are set as default value in Table 3.4.

few acceptable values are scattered over plot. We draw the value of $\tau_g = 10^{-7}$ and $\tau_l = 10^{-6}$ as default.

Except specifically mentioned, all settings are following default values in Table 3.2. We test with different S in Section 3.3.4 that directly increase the amount of training data at each round. N denotes the overall number of data samples that differ at different tasks. Local clients train autoencoders θ_l with local learning rate η_l and local adaptivity rate τ_l for E_l epochs. Subsequently, server learns θ_g with learning rate η_g with global adaptivity rate τ_g. Server also trains a classifier θ_c with learning rate η_c for E_c epochs. For choosing the best setting, we first conduct grid search for η_l, η_g, η_c based on grids: $\{10^{-4}, 10^{-3.5}, 10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}\}$. Second, we grid search for τ_g, τ_l based on grids: $\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Note that both τ is set to 10^{-5} when tuning learning rate. We compress input feature to half by default and we also test with different r^f in Section 3.3.5. r is the level of IIDness where we set $r = 0$ as default. We utilize bootstrap aggregating strategy to train our models with random batch sizes b_e and sequence lengths in range $(125, 256)$.

Table 3.2: Default values and descriptions in basic setting.

Location	Notation	Description	Default Value
Global Server	T	The number of communication round	100
	K	The number of clients	100
	S	The number of participated client at each round	10
	N	The number of overall data samples	N/A
	θ_g	Parameter of autoencodoer	N/A
	θ_c	Parameter of classifier	N/A
	E_c	The number of epoch for training classifier	5
	η_g	Global learning rate	0.03
	η_c	Classifier learning rate	0.03
	τ_g	Global training adaptivity	0.001
	r^f	The compression ratio of autoencoder	0.5
	r^l	The ratio of labeled data on server	0.1
Local Clients	E_l	Number of epoch for locally autoencoder training	1
	η_i	i-th local parameter of autoencoder	N/A
	η_l	Learning rate of client	0.03
	r	The level of IIDness	0
	τ_l	Local training adaptivity	0.001
Other	b_e	Batch size of training	(125, 256)

3.3.2 Comparison With Other Methods

Table 3.3: F1-score and accuracy of five baseline methods over three tasks. Boldface numbers indicate the best classification performance.

	OPP		DG		PAMP2	
Performance	F_m	Acc	F_m	Acc	F_m	Acc
FedSSL	0.282	0.824	0.510	0.726	0.230	0.478
FedGrad	0.308	0.834	0.493	0.788	0.278	0.521
AdaAlter	0.302	0.835	0.522	0.727	0.247	0.439
FedAdaGrad	0.285	0.841	0.566	0.772	0.329	0.536
AdaFedSSL	**0.386**	**0.862**	**0.637**	**0.872**	**0.408**	**0.633**
Delta from median	ΔF_m	ΔAcc	ΔF_m	ΔAcc	ΔF_m	ΔAcc
FedSSL	0.102	0.231	0.259	0.323	0.143	0.213
FedGrad	0.183	0.172	0.238	0.245	0.104	0.242
AdaAlter	**0.083**	**0.089**	0.201	0.218	0.094	0.193
FedAdaGrad	0.203	0.124	0.147	0.184	0.135	**0.133**
AdaFedSSL	0.093	0.127	**0.112**	**0.172**	**0.078**	0.136

We compare the performance of AdaFedSSL with four methods, *FedSSL*, *FedAdaGrad*, *AdaAlter* and *FedGrad*. The key difference between these methods is shown in Table 3.5. To measure

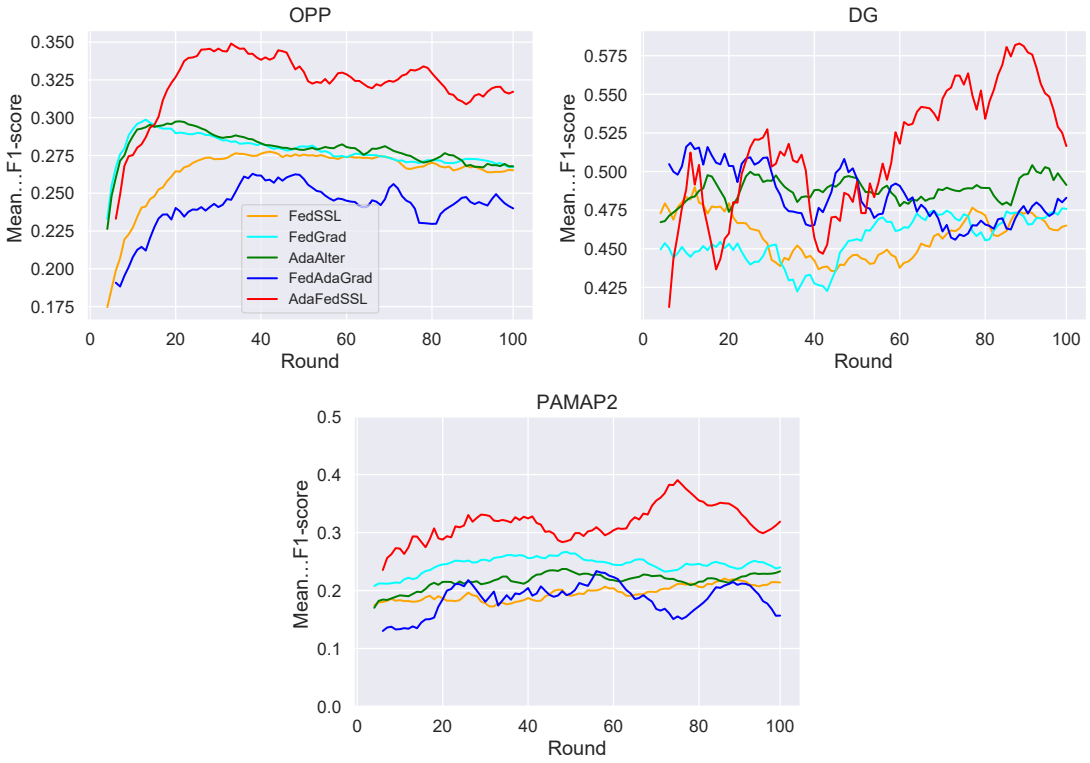

Figure 3.7: Learning curves of AdaFedSSL (Red) and other baseline methods over 100 training rounds.

the model performance, both test accuracy and mean F1-score are recorded. For best performance comparison, we conduct grid search of $\eta_g, \eta_c, \eta_l, \tau_l, \tau_g$ for each task and each method. The setting with best performance is in Table 3.4.

We plot the learning curves of mean F1-score over rounds in Figure 3.7. Each line is the average of 10 experiment replicas with a different seed. AdaFedSSL generates the highest score over other learning curves with few fluctuations during the process on three datasets. In the same way, the mean F1-score of FedAdaGrad floats up and down while FedSSL, FedGrad and AdaAlter stay smooth during training. All methods have comparable learning ability on DG while the gap increases when on OPP and PAMAP2, where AdaFedSSL show obvious superior performance over other methods. A clear trend that first goes up and then gradually declines can be drawn on OPP for five methods, while the score of our method surpasses others at the beginning 20 rounds and takes the lead till the end. FedAdaGrad shows the worst performance on this task. For DG, five lines stay close while AdaFedSSL takes effect during the last 40 rounds where its mean F1-score increases significantly. We can see the performance gap between five methods on PAMAP2 starts from the first round and continues until the end.

We quantify the learning performance by recording the highest mean F1-score and classification

Table 3.4: Hyperparameter setting for five methods (log_{10})

Dataset	Method	η_l	η_g	η_c	τ_l	τ_g
OPP	FedSSL	-2.5	0	-3	N/A	N/A
	FedGrad	-3.5	0	-3	-5	N/A
	AdaAlter	-3	0	-3	-6	N/A
	FedAdaGrad	-2.5	-2	-3	N/A	-4
	AdaFedSSLn	-1	-1.5	-3	-7	-6
DG	FedSSL	-3	0	-2	N/A	N/A
	FedGrad	-2	0	-2	-5	N/A
	AdaAlter	-2.5	0	-2	-4	N/A
	FedAdaGrad	-1	-1.5	-2	N/A	-4
	AdaFedSSLn	-2	-1	-2	-4	-6
PAMAP2	FedSSL	-2.5	0	-3	N/A	N/A
	FedGrad	-2.5	0	-3	-4	N/A
	AdaAlter	-3	0	-3	-4	N/A
	FedAdaGrad	-2.5	-2	-3	N/A	-4
	AdaFedSSLn	-2.5	-1	-3	-5	-5

Table 3.5: Key elements of baseline methods

Method	Local Optimizer	Server Optimizer	local gradient accumulator
FedSSL	SGD	SGD	N/A
FedGrad	AdaGrad	SGD	w/o synchronization
AdaAlter	AdaGrad	SGD	w/ synchronization
FedAdaGrad	SGD	AdaGrad	N/A
AdaFedSSLn	AdaGrad	AdaGrad	w/ synchronization

accuracy in Table 3.3. Our method outperforms all other methods on both metrics. Overall we obtain a very large performance improvement on PAMAP2 with more than 77% mean F1-score between AdaFedSSL and FedSSL, while on OPP and DG, this increase is smaller but still significant at 36% and 25%, respectively. The mean F1-scores of OPP and PAMAP2 are generally lower than that of DG (only having 3 classes), indicating magnificent unbalance between classes on these two tasks. Thus we put more attention on the accuracy of OPP and PAMAP2. Accuracy of classifying daily morning activity (OPP) has an insignificant spread of 0.03 between the best performance of AdaFedSSL and the worst of FedSSL. While for mean F1-score, the spread rises to 0.104 which demonstrates AdaFedSSL truly improves model performance by adding two accumulators. Similarly, the good performance of AdaFedSSL is also expressed by the score of DG and PAMP that improves 0.07 from the best of other methods on both tasks. One notable improvement is the accuracy from 0.788 of FedGrad to 0.872 of AdaFedGrad with more than 10% corresponding metric. FedGrad and AdaAlter perform equally on OPP and DG with a difference

of less than 6% mean F1-score. FedGrad is slightly better than AdaAlter on physical activity classification with an improvement of 10% mean F1-score. However, it shows unstable results but has better performance on OPP and PAMAP2 while worse result on DG. FedAdaGrad typically is the best method other than ours. It shows trivial degradation of 7% of OPP from FedGrad and light upgrade of 7% of DG from AdaAlter on mean F1-score, while boosts more than 17% score on PAMPA2 from FedSSL. Considering the performance fluctuation between experiment replicas, we also illustrate the variance at half bottom of Table 3.3. Effect of two adaptivity rates results in the lowest mean F1-score and accuracy variance on DG while comparable results on OPP and PAMAP2. We observe the model performance of AdaAlter and AdaFedSSL remains steady on OPP and PAMAP that only a small amount of fluctuation appears in mean F1-score. However, DG brings more than 17% mean F1-score and 19% accuracy variance on AdaFedSSL, which is still the smallest spread in both metrics. Combining the variation between tasks and experiment replicas, we argue that our method is stable than other baseline methods.

3.3.3 Update Accumulator to All Users or Participated Users?

Figure 3.8: **Learning curves of AdaFedSSL in two different updating techniques.** We compare two updating technique that are sharing average local accumulator to All Clients or only Participated Clients. Two scenarios are based on same hyperparameter configuration.

The difference between AdaAlter and FedGrad resides in how to dealing with local accumulators. In FedGrad, local clients accumulate past gradients without transferring them to central server, which means local gradients are not shared with other clients. AdaAlter remedies this by making clients sending local gradient accumulators to server, which shares the average of accumulators among clients selected at next round. This shared gradient breaks the isolated islands of information and provides an opportunity to make past gradients useful for other clients. However, there are two options when updating updated gradients. The first option is that server only broadcasts gradients to participated clients at current training round. In other words, those who contribute to updated gradients are eligible for updating their local accumulator. Essentially, this method moderates past gradient by taking average and only take effect on participated clients while not influence those who are not selected in training. The other option is that server broadcasts updated gradient to all clients. Intuitively, the later option may have issue with partial training as clients have large gradient accumulator at later round though it is the first time they are selected.

We plot the learning curves of two options in Figure 3.8 at the same way. For better comparison, we take window size of 5 to rolling average the values. Overall we observe a better performance of updating participated clients. Same with our intuition, for three tasks red line (updating all clients) shows a smoother process than green line (updating participated clients) because all clients sharing same accumulated gradient would make later training invalid. Partial update brings a higher F1-score while the violent fluctuation during training brings unpredictability. For example, the last round of PAMAP2 falls off more than 25% of that at the best round. Generally, for better model performance, we use partial updating at default.

3.3.4 Effect of S and r

We conduct experiments to test the effect of various statistical settings. We first compare model performance with different S, *i.e.*, different numbers of partial clients. The result is shown in the left of Figure 3.9. Mean F1-score of DG stays around 0.5 with a small amount of fluctuation. For OPP and PAMAP2, the undulation is more obvious than DG while remaining near 0.3 and 0.225, respectively. In general, there is no obvious relationship between S and mean F1-score. Essentially, with a higher S, training process is more similar to distributed SGD and having heavier communication burden (server broadcasting and receiving information from more clients). A higher S may reach global optima in a small number of rounds for reducing communication costs. While in case of only considering model performance, a small S would be a better choice for AdaFedSSL. The other concern is about IIDnsee r. We plot the relationship between model performance and r as the function of mean F1-score with r. As shown in the right of Figure 3.9,

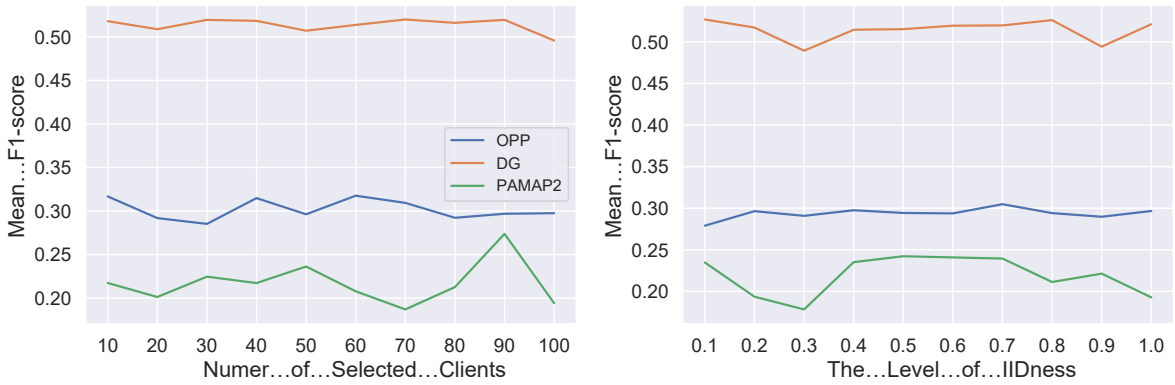

Figure 3.9: **Relationship between model performance and different statistical settings.** Left figure compares mean F1-score with different number of training clients. Right figure draw scores with different IIDness. Both experiments show stable performance produced by AdaFedSSL in inferior statistical setting.

with similar results in S, three parallel lines remain near steady when changing r. Except for PAMAP2, we see the score jumps up and down at the lower level of IIDness. Overall, we cannot see any advantage of increase the r to improve model performance. From these two experiments, we demonstrate stable performance of AdaFedSSL, even with extremely poor statistic situations such as only 10% of partial clients and the most severe Non-IID data.

3.3.5 Effect of Autoencoder

One may be curious about whether the compression rate of autoencoder takes effect in training a two-step HAR. As we define the dimensionality of encoded feature is the round value of the original number of feature dimensions multiply r^f, intuitively a lower r^f means more information from data is compressed. We compare model performance with $r^f \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ on three tasks. For each setting, we tune learning rates and adaptivity rates separately. The results are shown in Figure 3.10. Basically, $r^f = 0.1$ draws the worst learning process compared with others. For OPP, mean F1-score peaks at 0.2 and declines to almost zero during training. This rate hardly keeps data information resulting in invalid training. With regard to $r^f = 0.3, 0.5, 0.7, 0.9$, all experiments produce comparable results. In specific, setting of $r^f = 0.7$ generates highest mean F1-score while $r^f = 0.9$ has insignificant performance decrease compared with $r^f = 0.3$. An autoencoder partially filters useful information from original data and provides accurate latent representation for further classification. Results from DG are complicated. As it only has three classes and nine features, a sophisticated representation plays an instinctive role. Basically all five settings have nearly same training curves that the value of F1-score floats up and down till the end. Nevertheless, we can see a clear disadvantage of taking lower r^f in PAMAP2 and OPP. $r^f = 0.5$

Figure 3.10: **Relationship between model performance and compression rate.** Left figure shows mean F1-score on OPP where $r^f = 0.7$ generates the best result. Middle figure draws scores on DG where compression rate plays unimportant role in training. Right figure shows reaults of PAMAP2 where $r^f = 0.5$ is the best choice.

has the highest mean F1-score on PAMAP2 and $r^f = 0.9$ is the second. With half number of features, autoencoder sufficiently grasps core factors that determine the Parkinson symptom. Same with OPP, $r^f = 0.1$ has invalid results.

Another interesting element is the type of autoencoder. We compare three autoencoders of, *i.e.*, fully-connected (FC-AE), convolutions neural network (CNN-AE) and long short-term memory (LSTM-AE). All settings are tuned to fit the best learning rate and adaptivity rate. CNN-AE applies a 1-layer convolution to our one-channel input data. Followed by a batch normalization, ReLu activation and a fully connected layer, CNN-AE takes much more computation resources at local sides. With same compression rate, the number of parameters of CNN-AE is (51307, 22149, 872) of (OPP, PAMAP2, DG), respectively, while the number of FC only has (6439, 2782, 104). LSTM-AE has same structure as sever classifier defined in Section 3.2.2. However, instead of having a sizeable hidden size, we directly compress original features to hidden representations, *i.e.*, hidden size equals to the number of latent features. The results are shown in Figure 3.11. CNN-AE performs best on PAMAP2 while worst on DG as no complex feature needs to be extracted on

Figure 3.11: Relationship between model performance and model type.

DG. It has a comparable result with FC-AE on OPP that mean F1-score increases at first 30 rounds while declines at last 70 rounds. Interestingly, CNN-AE dominates the leaderboard at around round 20 due to a significant drop of FC-AE. On PAMAP2, a consistent increase can be seen while the other two stay flat. The advantage of CNN is that its capability to capture complex features makes it suitable for complicated data while at the cost of a heavier model. On both OPP and PAMAP2, LSTM-AE makes worst results and a slightly higher score than CNN-AE on DG, which means taking a small hidden size damages representation learning.

CHAPTER 4

Conclusions and Future Work

This thesis has presented two efficient federated learning approaches for image classification and human activity recognition. Both frameworks achieve superior performance compared with competing baseline methods. The key element contributing to the better performance of AdaFedSSL is adaptive optimizers utilized on both clients and server. AdaGrad employs an accumulator to record past gradients; thereby remembering the information learned from past data. The local accumulator lessens the adverse effects brought by heterogeneity among clients, while the global accumulator helps guarantee that the local optima do not overwhelm the global optima. For image classification, we weight the number of images with pseudo-labels instead of the number of all images in aggregation. Hyperparameters can be easily tuned in that a multitude of learning rates and adaptivity rates are suitable for different settings. We applied unsupervised learning on clients and supervised learning on the central server with the goal of maximizing the use of unlabeled data for human activity recognition. The local accumulator helps autoencoder training, while the global accumulator limits the large jump of model updates in the global autoencoder. We demonstrated through extensive experiments and ablation studies the superior performance of our proposed method in comparison with existing techniques in the literature. In a balanced setting, our method is robust to bad data distribution and partial participation, which are major statistical concerns in federated learning. In Section 4.1, the contributions made in each of the previous chapters and the concluding results drawn from the associated research work are presented. The limitations of the proposed approaches are discussed in Section 4.2. Suggestions for future research directions related to this thesis are also provided in Section 4.3.

4.1 Contributions of the Thesis

4.1.1 Adaptive Federated Semi-Supervised Learning for Image Classification

In Chapter 2, we proposed an adaptive federated semi-supervised learning framework that adopts adaptive optimizers on both client and server sides. We find that AdaFedSSL is robust and effective in both balanced and unbalanced settings. Our work can be easily implemented in smart cities systems, as we consider training in a realistic setting that takes stragglers, random labeling behaviors, client heterogeneity, and partial participation into account. We found that taking the number of samples after pseudo-labeling as weighting can effectively mitigate the adverse effects of unbalanced partition. Through our extensive experiments, we also found that AdaFedSSL can be easily tuned and is robust to various statistical issues. We also studied ways to reduce the communication cost of AdaFedSSL.

4.1.2 Adaptive Federated Semi-Supervised Learning for Human Activity Recognition

In Chapter 3, we introduced an adaptive federated learning approach for semi-supervised human activity classification on time series data by collaboratively learning an autoencoder and centrally learning a classifier. We incorporated AdaGrad optimizers into both local learning and global aggregation in a bid to make federated learning more stable under Non-IID data and partial training. We demonstrated the competitive and superior performance of AdaFedSSL in terms of mean F1-score over standard baseline methods on several benchmarks through extensive experiments. We also conducted ablation studies to assess the key elements such as local or global accumulators to better understand AdaFedSSL.

4.2 Limitations

While the proposed adaptive semi-supervised federated learning frameworks are capable of providing stable and superior training through adaptive optimizers, they do not, however, take communication efficiency into account. AdaFedSSL requires communicating both model update and client's accumulator that shares the same size with model trainable weights. Therefore, the communicating cost doubles relative to *FedAvg*. Due to similar reasons, our method also increases clients memory needed for computation, which may exceeds clients computational capability. For image classification, severe Non-IID data has a negative effect on training, and model performance depends heavily on the size and quality of data. For human activity recognition, though data is

easier to collect than image data, preprocessing such as slicing and feature extraction is relatively difficult. Also, how to use labeled data on clients is quite challenging.

4.3 Future Work

Several interesting research directions, motivated by this thesis, are discussed below:

4.3.1 Statistical Concerns

In a much larger scale, statistical concerns magnify the importance of stability in federated learning. We aim to explore alternative algorithms that are stable with severe Non-IID data and imbalanced data.

4.3.2 Semi-Supervised Learning on Clients

For time series data, using proxy-label methods to generate pseudo-label to increase the model performance is quite promising. In a real scenario, local data could be labeled by users, and semi-supervised training on clients would be of great interest.

4.3.3 Balancing Communication Burden and Model Performance

Communication bottlenecks impede the development of federated learning as the number of training round increases when model converges slowly, or the amount of communication increases when transferring complicated messages needed for advanced algorithms. How to balance the communication burden and training efficiency is of paramount importance in federated learning. For future work, we plan to investigate more adaptive optimizers, and at the same time not exceeding the communication bottleneck.

References

[1] Z. Long, L. Che, Y. Wang, M. Ye, J. Luo, J. Wu, H. Xiao, and F. Ma, "Fedsemi: An adaptive federated semi-supervised learning framework," *arXiv preprint arXiv:2012.03292*, 2020.

[2] X. Peng, Z. Huang, Y. Zhu, and K. Saenko, "Federated adversarial domain adaptation," *arXiv preprint arXiv:1911.02054*, 2019.

[3] 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.

[4] 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.

[5] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[6] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

[7] 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.

[8] X. Yao, T. Huang, C. Wu, R.-X. Zhang, and L. Sun, "Federated learning with additional mechanisms on clients to reduce communication costs," *arXiv preprint arXiv:1908.05891*, 2019.

[9] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," in *International Conference on Machine Learning*, pp. 8253–8265, PMLR, 2020.

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

[11] W.-T. Chang and R. Tandon, "Communication efficient federated learning over multiple access channels," *arXiv preprint arXiv:2001.08737*, 2020.

[12] M. Ribero and H. Vikalo, "Communication-efficient federated learning via optimal client sampling," *arXiv preprint arXiv:2007.15197*, 2020.

[13] Z. Zhang, Z. Yao, Y. Yang, Y. Yan, J. E. Gonzalez, and M. W. Mahoney, "Benchmarking semi-supervised federated learning," *arXiv preprint arXiv:2008.11364*, 2020.

[14] W. Jeong, J. Yoon, E. Yang, and S. J. Hwang, "Federated semi-supervised learning with inter-client consistency," *arXiv preprint arXiv:2006.12097*, 2020.

[15] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *CoRR*, vol. abs/1610.02527, 2016.

[16] 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*, pp. 1273–1282, PMLR, 2017.

[17] 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.

[18] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 1103–1111, IEEE, 2018.

[19] L. Zhou, S. El Helou, L. Moccozet, L. Opprecht, O. Benkacem, C. Salzmann, and D. Gillet, "A federated recommender system for online learning environments," in *International Conference on Web-Based Learning*, pp. 89–98, Springer, 2012.

[20] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[21] J. Wang, H. Liang, and G. Joshi, "Overlap local-sgd: An algorithmic approach to hide communication delays in distributed sgd," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8871–8875, IEEE, 2020.

[22] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, pp. 4424–4434, 2017.

[23] P. P. Liang, T. Liu, L. Ziyin, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global representations," *arXiv preprint arXiv:2001.01523*, 2020.

[24] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, "Atomo: Communication-efficient learning via atomic sparsification," in *Advances in Neural Information Processing Systems*, pp. 9850–9861, 2018.

[25] X. Yao, T. Huang, C. Wu, R.-X. Zhang, and L. Sun, "Federated learning with additional mechanisms on clients to reduce communication costs," *arXiv preprint arXiv:1908.05891*, 2019.

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

[27] W.-T. Chang and R. Tandon, "Communication efficient federated learning over multiple access channels," *arXiv preprint arXiv:2001.08737*, 2020.

[28] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," in *International Conference on Machine Learning*, pp. 8253–8265, PMLR, 2020.

[29] X. J. Zhu, "Semi-supervised learning literature survey," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2005.

[30] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[31] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," in *Advances in Neural Information Processing Systems*, pp. 5049–5059, 2019.

[32] K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel, "Fixmatch: Simplifying semi-supervised learning with consistency and confidence," *arXiv preprint arXiv:2001.07685*, 2020.

[33] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10687–10698, 2020.

[34] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *Advances in neural information processing systems*, pp. 1163–1171, 2016.

[35] G. French, M. Mackiewicz, and M. Fisher, "Self-ensembling for visual domain adaptation," *arXiv preprint arXiv:1706.05208*, 2017.

[36] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 1979–1993, 2018.

[37] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov, "Good semi-supervised learning that requires a bad gan," in *Advances in neural information processing systems*, pp. 6510–6520, 2017.

[38] G. J. McLachlan, "Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis," *Journal of the American Statistical Association*, vol. 70, no. 350, pp. 365–369, 1975.

[39] E. Arazo, D. Ortego, P. Albert, N. E. O'Connor, and K. McGuinness, "Pseudo-labeling and confirmation bias in deep semi-supervised learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[40] H. Pham and Q. V. Le, "Semi-supervised learning by coaching," 2019.

[41] Y. Zhao, H. Liu, H. Li, P. Barnaghi, and H. Haddadi, "Semi-supervised federated learning for activity recognition," *arXiv preprint arXiv:2011.00851*, 2020.

[42] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto, "Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data," *arXiv preprint arXiv:2008.06180*, 2020.

[43] Y. Kang, Y. Liu, and T. Chen, "Fedmvt: Semi-supervised vertical federated learning with multiview training," *arXiv preprint arXiv:2008.10838*, 2020.

[44] Y. Jin, X. Wei, Y. Liu, and Q. Yang, "A survey towards federated semi-supervised learning," *arXiv preprint arXiv:2002.11545*, 2020.

[45] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[46] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, "Secureboost: A lossless federated learning framework," *arXiv preprint arXiv:1901.08755*, 2019.

[47] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent.," in *NIPS*, vol. 4, p. 4, Citeseer, 2010.

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

[49] C. Xie, O. Koyejo, I. Gupta, and H. Lin, "Local adaalter: Communication-efficient stochastic gradient descent with adaptive learning rates," *arXiv preprint arXiv:1911.09030*, 2019.

[50] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[51] Q. Tong, G. Liang, and J. Bi, "Effective federated adaptive gradient methods with non-iid decentralized data," *arXiv preprint arXiv:2009.06557*, 2020.

[52] N. Y. Hammerla, S. Halloran, and T. Plötz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," *arXiv preprint arXiv:1604.08880*, 2016.

[53] A. Bulling, U. Blanke, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

[54] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *Aaai*, vol. 5, pp. 1541–1546, Pittsburgh, PA, 2005.

[55] M. Zhang and A. A. Sawchuk, "Motion primitive-based human activity recognition using a bag-of-features approach," in *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, pp. 631–640, 2012.

[56] A. Wickramasinghe, D. C. Ranasinghe, C. Fumeaux, K. D. Hill, and R. Visvanathan, "Sequence learning with passive rfid sensors for real-time bed-egress recognition in older people," *IEEE journal of biomedical and health informatics*, vol. 21, no. 4, pp. 917–929, 2016.

[57] T. Plötz, N. Y. Hammerla, and P. L. Olivier, "Feature learning for activity recognition in ubiquitous computing," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[58] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[59] J. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[60] R. Yao, G. Lin, Q. Shi, and D. C. Ranasinghe, "Efficient dense labelling of human activity sequences from wearables using fully convolutional networks," *Pattern Recognition*, vol. 78, pp. 252–266, 2018.

[61] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.

[62] C. Bettini, G. Civitarese, and R. Presotto, "Personalized semi-supervised federated learning for human activity recognition," *arXiv preprint arXiv:2104.08094*, 2021.

[63] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent iot applications: A cloud-edge based framework," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 35–44, 2020.

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

[65] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," *arXiv preprint arXiv:1910.03581*, 2019.

[66] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, "Pmf: A privacy-preserving human mobility prediction framework via federated learning," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–21, 2020.

[67] Q. Wu, X. Chen, Z. Zhou, and J. Zhang, "Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring," *IEEE Transactions on Mobile Computing*, 2020.

[68] Y. Zhao, H. Haddadi, S. Skillman, S. Enshaeifar, and P. Barnaghi, "Privacy-preserving activity and health monitoring on databox," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 49–54, 2020.

[69] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[70] K. M. Rashid and J. Louis, "Times-series data augmentation and deep learning for construction equipment activity recognition," *Advanced Engineering Informatics*, vol. 42, p. 100944, 2019.

[71] J. Wang, Y. Chen, Y. Gu, Y. Xiao, and H. Pan, "Sensorygans: An effective generative adversarial framework for sensor-based human activity recognition," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2018.

[72] M. H. Chan and M. H. M. Noor, "A unified generative model using generative adversarial network for activity recognition," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2020.

[73] B. Longstaff, S. Reddy, and D. Estrin, "Improving activity classification for health applications on mobile devices using active and semi-supervised learning," in *2010 4th International Conference on Pervasive Computing Technologies for Healthcare*, pp. 1–7, IEEE, 2010.

[74] Y.-S. Lee and S.-B. Cho, "Activity recognition with android phone using mixture-of-experts co-trained with labeled and unlabeled data," *Neurocomputing*, vol. 126, pp. 106–115, 2014.

[75] M. Stikic, D. Larlus, S. Ebert, and B. Schiele, "Weakly supervised recognition of daily life activities with wearable sensors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 12, pp. 2521–2537, 2011.

[76] V. Hernandez, D. Kulić, and G. Venture, "Adversarial autoencoder for visualization and classification of human activity: Application to a low-cost commercial force plate," *Journal of biomechanics*, vol. 103, p. 109684, 2020.

[77] B. Almaslukh, J. AlMuhtadi, and A. Artoli, "An effective deep autoencoder approach for online smartphone-based human activity recognition," *Int. J. Comput. Sci. Netw. Secur*, vol. 17, no. 4, pp. 160–165, 2017.

[78] A. A. Varamin, E. Abbasnejad, Q. Shi, D. C. Ranasinghe, and H. Rezatofighi, "Deep auto-set: A deep auto-encoder-set network for activity recognition using wearables," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 246–253, 2018.

[79] F. Gu, K. Khoshelham, S. Valaee, J. Shang, and R. Zhang, "Locomotion activity recognition using stacked denoising autoencoders," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2085–2093, 2018.

[80] X. Gao, H. Luo, Q. Wang, F. Zhao, L. Ye, and Y. Zhang, "A human activity recognition algorithm based on stacking denoising autoencoder and lightgbm," *Sensors*, vol. 19, no. 4, p. 947, 2019.

[81] W. Wei, H. Wu, and H. Ma, "An autoencoder and lstm-based traffic flow prediction method," *Sensors*, vol. 19, no. 13, p. 2946, 2019.

[82] H. Zou, Y. Zhou, J. Yang, H. Jiang, L. Xie, and C. J. Spanos, "Deepsense: Device-free human activity recognition via autoencoder long-term recurrent convolutional network," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

[83] Y. Li, D. Shi, B. Ding, and D. Liu, "Unsupervised feature learning for human activity recognition using smartphone sensors," in *Mining intelligence and knowledge exploration*, pp. 99–107, Springer, 2014.

[84] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, *et al.*, "Citypulse: Large scale data analytics framework for smart cities," *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[85] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 88–93, 2020.

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

[87] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[88] A. Albaseer, B. S. Ciftler, M. Abdallah, and A. Al-Fuqaha, "Exploiting unlabeled data in smart cities using federated edge learning," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 1666–1671, IEEE, 2020.

[89] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

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

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

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

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

[94] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[95] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

[96] 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.

[97] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan, *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[98] Y. Cao, P. Hou, D. Brown, J. Wang, and S. Chen, "Distributed analytics and edge intelligence: Pervasive health monitoring at the era of fog computing," in *Proceedings of the 2015 Workshop on Mobile Big Data*, pp. 43–48, 2015.

[99] S. Enshaeifar, A. Zoha, A. Markides, S. Skillman, S. T. Acton, T. Elsaleh, M. Hassanpour, A. Ahrabian, M. Kenny, S. Klein, *et al.*, "Health management and pattern analysis of daily living activities of people with dementia using in-home sensors and machine learning techniques," *PloS one*, vol. 13, no. 5, p. e0195605, 2018.

[100] J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Edge-ai in lora-based health monitoring: Fall detection system with fog computing and lstm recurrent neural networks," in *2019 42nd international conference on telecommunications and signal processing (TSP)*, pp. 601–604, IEEE, 2019.

[101] S. Enshaeifar, P. Barnaghi, S. Skillman, A. Markides, T. Elsaleh, S. T. Acton, R. Nilforooshan, and H. Rostill, "The internet of things for dementia care," *IEEE Internet Computing*, vol. 22, no. 1, pp. 8–17, 2018.

[102] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, JMLR Workshop and Conference Proceedings, 2012.

[103] S. Singh, S. Bhardwaj, H. Pandey, and G. Beniwal, "Anomaly detection using federated learning," in *Proceedings of International Conference on Artificial Intelligence and Applications*, pp. 141–148, Springer, 2021.

[104] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[105] B. van Berlo, A. Saeed, and T. Ozcelebi, "Towards federated unsupervised representation learning," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 31–36, 2020.

[106] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[107] A. Gensler, J. Henze, B. Sick, and N. Raabe, "Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks," in *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 002858–002865, IEEE, 2016.

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

[109] J. Chen, X. Qiu, P. Liu, and X. Huang, "Meta multi-task learning for sequence modeling," *Neurocomputing*, 2018.

[110] R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. d. R. Millán, and D. Roggen, "The opportunity challenge: A benchmark database for on-body sensor-based activity recognition," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 2033–2042, 2013.

[111] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th international symposium on wearable computers*, pp. 108–109, IEEE, 2012.

[112] M. Bächlin, D. Roggen, G. Tröster, M. Plotnik, N. Inbar, I. Maidan, T. Herman, M. Brozgol, E. Shaviv, N. Giladi, *et al.*, "Potentials of enhanced context awareness in wearable assistants for parkinson's disease patients with the freezing of gait syndrome.," in *ISWC*, pp. 123–130, 2009.