FedPA: an Adaptively Partial Model Aggregation Strategy in Federated Learning

Juncai Liu[a], Jessie Hui Wang[a], Chenghao Rong[a], Yuedong Xu[b], Tao Yu[a], Jilong Wang[a]

[a]*Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University*
[b]*School of Information Science and Engineering, Fudan University*

Abstract

Federated Learning has sparked increasing interest as a promising approach to utilize large amounts of data stored on network edge devices. Federated Averaging is the most widely accepted Federated Learning framework. In Federated Averaging, the server keeps waiting for client models to compute the global model in each round unless all client models are received or a pre-configured timer expires, therefore it suffers seriously from participant devices with weak computation and/or communication capability, which is a kind of straggler problem. In this paper we design FedPA, a framework based on partial model aggregation strategy, in which the server waits for only an appropriate number of device models (referred to as aggregation number) in each round. Our experiment shows that the accuracy loss of the aggregated global model in a single round is not significant if the aggregation number is decided carefully. We propose a waiting strategy to determine the aggregation number for each round dynamically and the aggregation number is adaptive to achieve a tradeoff between single-round training time and the expected number of rounds to reach the target accuracy. Stale models are also included during aggregation when they arrive, and their positive value and negative effect are carefully evaluated and reflected in the aggregation strategy. Experiments show that FedPA outperforms the baseline strategy FedAvg and other three algorithms named FedAsync, FLANP and AD-SG. It can work well in all scenarios with different distributions of data samples (characterized by non-IID ratio) and computation/communication capability (characterized by level of heterogeneity) among devices. Experiments also show that FedPA is robust when a certain amount of noise is added into the input from clients for privacy concerns.

Keywords: Federated Learning, Aggregation Strategy, Straggler Problem

1. Introduction

It was reported that there had been 3 billion active smartphones [1] and nearly 7 billion connected Internet of Things (IoT) devices [2] all over the world, and the numbers kept increasing in these years. These devices continuously generate large volumes of data, *e.g.*, photos, voice, keystrokes, which are valuable for machine learning applications to train better models. By using data from mobile devices or IoT devices, some machine learning applications, such as image or video classification [3][4], speech recognition [5], and Smart Keyboard [6], can significantly improve the experience of users of these services. Meanwhile, edge devices are equipped with more powerful computation capability than before. It is expected that AI components will be embedded in most end devices in the near future [7][8], which means that these end devices can participate in model training tasks and complete some computation jobs.

Federated Learning [9][10] is a model training framework in which each end device uses its own data and computation capability to train its local device model and a shared global model is generated by aggregating all local device models submitted by the participant devices in the training task. As shown in Figure 1, in each round, the server randomly selects a subset of clients and distributes model parameters to the selected clients (Step 1). These clients update the model locally using their local data and get their individual device models (Step 2). It is required that these devices should upload their device models to the server (Step 3). The server waits for these device models and aggregates the models received to obtain a global model to complete a round (Step 4). The above procedures repeat for a lot of rounds until the accuracy of the global model becomes satisfactory.

Federated learning is a promising model training framework. However, it still faces some challenges. End devices are heterogenous and unstable. They are not dedicated to model training tasks. Therefore their performance and availability cannot be guaranteed during training. Some end devices are equipped with powerful computing capabilities and larger network bandwidth, and they can complete their local training very fast. Some devices are less powerful in computation and/or communication, and their device models will arrive at the server very late. Furthermore, some devices may go offline during training due to a lack of battery power or an unreliable network connection,

*Corresponding author
Email address: jessiewang@tsinghua.edu.cn (Jessie Hui Wang)

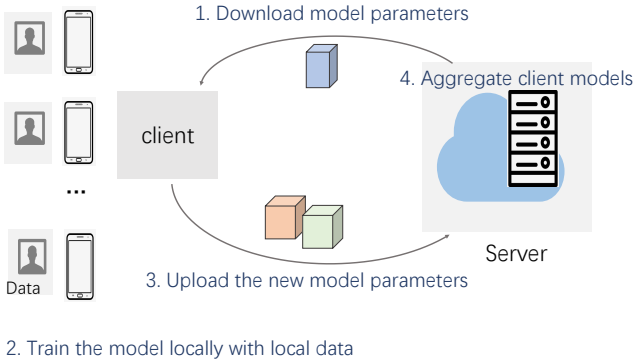


Figure 1: Federated Learning

and their device models will never arrive at the server.

In Federated Learning, Federated Averaging (FedAvg) [9] is the most widely accepted algorithm, in which the server keeps waiting for client models to compute the global model unless all client models have been received or a pre-configured timer expires. Obviously, if one or some clients are very slow, the server and other devices have to be idle for a long period, which increases the training time. These slow clients are referred to as *stragglers* and the relevant problem is named as *straggler problem*.

In FedAvg, all participants are required to be synchronous, which is the reason for the straggler problem. Researchers try to propose asynchronous [11] or semi-synchronous [12] methods to mitigate or eliminate the straggler problem. However, these methods converge slowly when the data samples among end devices are unbalanced and non-IID (identically and independently distributed) [13][14][15][16].

Intuitively, if the server aggregates only the client models from fast devices in a round and starts the next round without further waiting for client models from slow devices, the training time for this single round would be shortened. However, the training accuracy of the aggregated global model in this round is in doubt, and it may require more rounds to reach a satisfactory accuracy.

We conduct experiments to see the resulting accuracy when the server only aggregates a part of participants in a round. We refer to it as *partial model aggregation strategy*, and the number of client models used by the server is referred to as *aggregation number* of this round. Experiments show that the accuracy loss is not significant if the aggregation number is decided carefully. We further notice that the accuracy of the global model under our partial aggregation strategy is affected by the distribution of data samples among devices. Roughly speaking, the accuracy loss tends to be larger under the same aggregation number when the devices are more different in terms of the data samples owned by them. We can see that how to determine the aggregation number in each round is the key problem for the performance of the partial model aggregation strategy.

Based on the above observations from the experiments, we design and implement FedPA (Federated Partial Aggregation), a framework aiming to improve the efficiency of federated learning through a partial model aggregation strategy. FedPA uses reinforcement learning agent to adaptively determine the appropriate aggregation number in each round, and the aggregation number is adaptive to the data distribution of the selected devices and the heterogeneity of device capability. By making the aggregation number in each round adaptive and dynamic, it achieves a tradeoff between the training time taken by a single round and the expected number of rounds to reach the target accuracy.

A stale model arrives later but obviously it may have valuable information to improve the accuracy of the global model in the round, especially when the local models from this device have never been aggregated in global models of earlier rounds. On the other hand, stale models are trained from expired global models, which means placing too much emphasis on stale models can have negative influence on the convergence of the training process. Therefore, FedPA utilizes stale models instead of discarding them, but weighting factors are used to reflect the value of stale models. We propose an aggregation strategy which includes both fresh models and stale models and the weighted factor is determined based on the staleness of stale models and the number of data samples they represent.

We have implemented FedPA using TensorFlow. We conduct experiments to explore the influence factors of the output of the reinforcement learning agent and the robustness of FedPA when a certain amount of noise is added into the input for privacy concerns. We evaluate its performance by conducting various federated learning tasks. Our experimental results show that our strategy outperforms traditional Federated Averaging strategy and other three algorithms called FedAsync, FLANP and AD-SGD on the CIFAR-10 dataset and the MINIST dataset. Furthermore, FedPA can work well in all scenarios with different distributions of data samples (characterized by non-IID ratio) and computation/communication capability (characterized by level of heterogeneity) among devices.

The rest of this paper is organized as follows. Section 2 summarizes related works on improving the efficiency of federated learning. In Section 3, we introduce a preliminary study on partial model aggregation, which shows the resulting accuracy when the server only aggregates a part of participants in a round and shows the influence factors to the accuracy of partial aggregation model. In Section 4, we detail the design of our FedPA framework. We present and discuss the experimental results in Section 5, and conclude this paper in Section 6.

2. Related Work

Reducing training time is an important research goal in federated learning and has attracted attention from many researchers. Parameter synchronization (Step 1 and 3 in

Figure 1) is one important bottleneck to achieve this goal. Generally said, there are two problems that significantly affect the time overhead of parameter synchronization, *i.e.*, *communication efficiency problem* and *straggler problem*.

2.1. Improving the communication efficiency of parameter synchronization

Devices in federated learning are assumed to be located at the edge of the Internet, and the communication resource between a device and the server is often very limited and should be used efficiently. Therefore, improving the communication efficiency of parameter synchronization is recognized as an important research problem and many researchers try to accelerate the training process by reducing the data volume of the communication for parameter synchronization.

For example, some works have proposed to compress the trained models and then the size of parameters to be synchronized is reduced [17][18][19][20][21]. The authors of [22] and [23] proposed to have clients perform multiple epochs (instead of a single epoch) of local training in each individual round of global aggregation. In this way, the frequency of parameter synchronization is reduced and thus the data volume of the communication is reduced. The authors of [24] proposed that the communication caused by insignificant updates should be eliminated to reduce the communication overhead. In their solution, the gradient updates of the local model should be accumulated, and an updated device model is sent out until the accumulation exceeds a preset "significance threshold".

2.2. Straggler problem

Straggler problem is a conventional problem in the area of parallel computing and has been noticed in distributed learning systems [25][26][27][28][29]. For example, an *asynchronous* solution is proposed in [26]. In this solution, the server updates the global model immediately once a single client model is collected. The updated global model is distributed to the client that sends the client model, and then the client can continue for the next round. In this way, fast clients do not need to wait for slow devices, and the straggler problem can be mitigated theoretically, but the training process may converge slowly or even develop incorrectly if the data distribution among devices is not independent and identical.

The straggler problem can be more serious in federated learning with massive mobile devices. In this scenario, the performance difference between fast mobile devices and slow mobile devices can be very large (tends to be much larger than the difference between two virtual machines in a data center), which means that fast devices need to wait for a longer period of time. The data distribution among devices is not independent and identical in this scenario [13][2], which brings challenges to asynchronous methods.

There are some research works on straggler problem in federated learning with mobile devices. There are mainly two ideas. The first one is to avoid selecting slow nodes (stragglers) in each round. The second is that the server does not wait for slow nodes in a round before entering the next round and stale models from stragglers can be aggregated into the global model when they arrive later.

As for the first idea, Nishio et.al [30] proposed FedCS, a protocol that estimates the completion time of each device and filters out slow devices in the client selection phase. This method has a potential risk that the final global model has serious bias and overfits to the data in the devices with high performance. FLANP, proposed by Reisizadeh et.al [31], sorts client nodes from fast to slow according to their completion time. It uses first several fast nodes for warm-up training, and gradually doubles the number of participants in the training process until the model reaches a satisfactory accuracy. FLANP always chooses participant nodes from fast to slow in each round and waits for all participants to return results before entering the next round. Thus, the global model at the warm-up phase has potential to overfit to the data in those fast nodes in non-IID scenarios and FLANP may suffer from stragglers when the number of participants is large.

As for the second idea, FedAsync, proposed by Xie et.al [11], is a total-asynchronous method. Similar to [26], this method also converges very slowly (even cannot converge), especially when the data distribution among devices is not independent and identical. SAFA, proposed by Wu et.al [12], asks all clients to train local models in each round and do not wait for those straggler nodes in each round. It aggregates normal and stale models without considering the negative impact of stale models. SAFA assumes that the client devices have sufficient energy and computation resources. In this paper, we believe it is more practical and reasonable that mobile devices have limited computation and energy resources. Therefore, in each round, our method only requires a sampled subset of clients to train local models, which is similar to FedAvg [9]. AD-SGD, proposed by Li et.al [32], aggregates normal and stale models to speed up convergence. The AD-SGD algorithm needs Hessian approximation matrixes to mitigate the negative impact of stale models. Therefore, this algorithm requires clients to transmit model parameters and gradient parameters to the server in each round, which means the volume of transmitted data is 2X of FedAvg and FedPA. In the WAN scenario, bandwidth resources are scarce and data transmission is time-consuming. Transmitting more data means more time needs to be spent on transmission.

3. Preliminary Study on Partial Model Aggregation

FedAvg suffers from stragglers seriously and each single round is likely to take a long time because it tries to aggregate models from all participants in each individual round. At the other extreme, *FedAsync* updates the global model once it receives a single device model. A single round is

completed very fast but it has a slow convergence rate, *i.e.*, it tends to need a lot of rounds to complete the total training successfully.

It is natural to ask whether we can aggregate models from a part of participants in each round and start the next round without further waiting for more client models. We refer to it as *partial model aggregation*. The key problem in this approach is how many models shall be waited before entering the next round.

In this section, we first present the concept of partial model aggregation mathematically. Then we conduct experiments to see the results of partial aggregation in a single round and explore the factors that have influence on the selection of aggregation number, which helps the subsequent method design.

3.1. The concept of partial model aggregation

The goal of machine learning is to solve the following equation to get the optimal model parameters w^*,

$$w^* = arg\min_{w} \frac{1}{|D|} \sum_{(x_i,y_i)\in D} f(w; x_i, y_i), \tag{1}$$

where w is the parameter of the model, D is the set of data samples, and $f(w; x_i, y_i)$ is the loss function, representing the loss of inference on data sample (x_i, y_i) by model w.

In federated learning, each device k ($k \in [1, N]$) has a subset of data samples, denoted by D_k. Using data samples in D_k, each device can train a local device model w^{k*}. As devices have different data samples, the device models they get would be different. The server aggregates these models to generate a global model and expect the global model can approximately solve the Equation 1. To improve the optimality of the solution achieved from aggregation, the local training and global aggregation are conducted for a lot of rounds. Mathematically, the process of conventional federated learning algorithm can be described as follows.

In each round t, the server randomly selects a subset C_t of devices to participate, where $|C_t| < N$. Each selected device k receives the aggregated model of the last round (denoted by w_{t-1}) from the server and train an updated local model w_t^k using Stochastic Gradient Descent (SGD) algorithm as follows. Note that each device can perform training of multiple epochs E configured by the algorithm. We represent $w_t^{k,\tau}$ as the model trained at τ-th epoch. Intuitively, $w_t^{k,0}$ is the model each device receives, *i.e.* w_{t-1} and $w_t^{k,E}$ is the updated local model w_t^k.

local training:

$$w_t^{k,\tau} = w_t^{k,\tau-1} - \eta \frac{\sum_{(x_i,y_i)\in D_k} \bigtriangledown f(w_t^{k,\tau-1}; x_i, y_i)}{|D_k|}, \tau \in [1, E]$$

where η is the configured learning rate of the algorithm.

The server waits for the local models from these devices $k \in C_t$, and aggregates them to obtain an updated global model of round t. Mathematically, we denote the global aggregation as follows.

global aggregation (all participants):

$$w_t = \mathcal{A}([w_t^k \ \forall k \in C_t])$$

To mitigate the impact of stragglers, *partial aggregation* strategy waits for and aggregates only a part of device models that arrive early. Mathematically, let $S(C_t, m)$ be the set of the earliest m devices in the set of selected devices C_t, and in partial aggregation we have

global aggregation (part of participants):

$$w_t(m_t) = \mathcal{A}([w_t^k \ \forall k \in S(C_t, m_t)])$$

where m_t is the number of device models that should take part in the global aggregation and we define the model $w_t(m_t)$ as m_t-aggregated model. Please note that $C_t = S(C_t, |C_t|)$ and $w_t = w_t(|C_t|)$ and we define the model $w_t(|C_t|)$ as *full aggregated model*.

For each round t, if we can find a m_t smaller than $|C_t|$ and $w_t(m_t)$ does not lose much precision, the concept of partial aggregation might be a feasible solution to the straggler problem.

The distribution of data samples among devices may have an influence on the precision of $w_t(m_t)$ and the significance of each individual device model. Intuitively, if a device has a special set of data samples (*i.e.*, other devices do not have), its device model might be very important for reducing deviation of the aggregated model. As for the distribution of training data, data is said to be independent and identical distribution (IID) if it is sampled independently from the same joint probability distribution of data features and labels. All devices will have roughly the same percentage of data of each label if the training data is IID. Otherwise, it is said to be non-IID.

In order to evaluate the precision of $w_t(m_t)$, we compare it to a "benchmark", *i.e.*, the full aggregated model $w_t(|C_t|)$. Let $\mathcal{D}(w_1, w_2)$ denote the distance between two models and we use the *Manhattan distance* to calculate $\mathcal{D}$. Mathematically we have

$$\mathcal{D}(w_1, w_2) = ||w_1 - w_2||.$$

Then the **deviation of** $w_t(m)$ can be denoted by

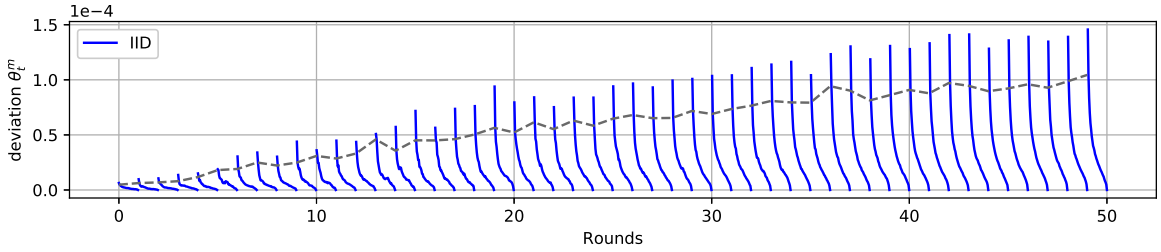
$$\theta_t^m = \mathcal{D}(w_t(m), w_t(|C_t|)),$$

which is the distance between the $m-$aggregated global model in round t and the global model aggregated from all participants in this round.

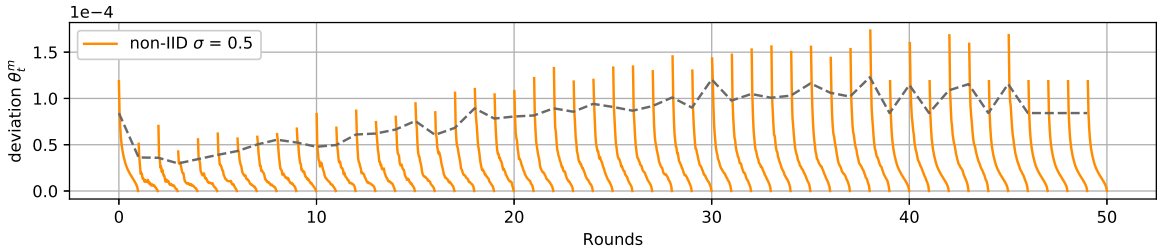
In the following subsections, we conduct experiments to see how the deviation of $m-$aggregated model (*i.e.*, θ_t^m) changes in each round t and the impact of unbalanced data distribution on the deviation.

3.2. Experiment settings

In the experiments, we use TensorFlow to train a two-layer CNN [33] model on the CIFAR-10 dataset. There



(a) IID



(b) non-IID

Figure 2: Deviation of partial aggregated model θ_t^m for different t and m (distance between the $m-$aggregated model and the global model aggregated from all participants in each round t). The gray line is θ_t^2 for different t.

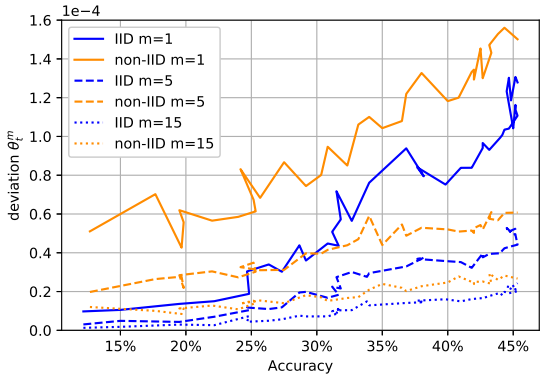


Figure 3: Deviation θ_t^m for different accuracy of aggregated models $w_t(|C_t|)$

are 100 devices in total, and each device is simulated by a thread. The local training and global aggregation are repeated for 50 rounds, *i.e.*, $t \in [1, 50]$. In each round t, the server randomly selects 30 clients to participate, *i.e.*, $|C_t| = 30$. Our server exploits FedAvg, a commonly used aggregation method in federated learning, to aggregate device models to obtain a partially-aggregated or full-aggregated global model.

As for IID and non-IID settings, we use a similar approach to [34] to generate data samples for each device. In the case of IID setting, data samples are randomly allocated to 100 devices and thus data samples of all devices have similar distribution among the 10 labels. In the case of non-IID setting, for each label, we retrieve σ $(0.1 < \sigma \leq 1)$ percentage of data samples with this label and evenly assign them to 10 random devices. The remaining samples with this label are assigned to the other 90 devices randomly. We repeat this process for all 10 labels. In this way, the selected 10 devices tend to have more samples with this label than other devices, and they also tend to have more samples with this label than other labels in the same device. σ can be tuned to control the degree of non-IID, and we name this parameter as *non-IID ratio*. As σ increases, it is more likely for an individual device model to be significant for reducing deviation of the global aggregated model. We conduct experiments for two scenarios, *i.e.*, IID and non-IID ($\sigma = 0.5$) respectively.

3.3. Experiment results

3.3.1. Deviation of partial aggregated model

We plot the deviation of partial aggregated model θ_t^m for different t and m in Figure 2. The x axis is labeled with the round index t ($t \in [1, 50]$). Please remind that $|C_t| = 30$ for all t. Therefore there are 30 points in a single round and each point represents a θ_t^m ($m \in [1, 30]$. We skip m in the label of x axis for brevity). From Figure 2, we can see that the deviation of partial aggregated model θ_t^m is large at the beginning but decreases quickly as m increases in each round under both scenarios of IID and non-IID, which means *the first some device models have been able to derive a relatively good aggregated model*. It can be seen that the contribution of each later device model to reduce the deviation of the aggregated model is small and negligible in almost all rounds.

3.3.2. Deviation caused by partial aggregation under different non-IID ratio

As we have mentioned, intuitively the partial aggregation may cause a larger deviation in a scenario with a larger non-IID ratio. We would like to examine whether this intuition is true.

The experiments with different non-IID ratios take different number of rounds to converge at a satisfactory accuracy. To enable the comparison between training tasks with different convergence rates, we use *model accuracy* (the accuracy of the full aggregated model $w_t(|C_t|)$ on the validation set) instead of the index of round (t) to align the rounds for comparison. For example, we compare the 12-th round of the IID scenario with the 15−th round of the non-IID scenario with $\sigma = 0.5$ since they can reach the approximately same model accuracy (30%) in this experiment. It can be viewed as that we use model accuracy as an indicator of the progress percentage of a training task and we define the accuracy as *training progress percentage*. We compare the deviation when the two scenarios have the same progress percentage.

We plot the deviation θ_t^m and the accuracy of the global model in the round t in Figure 3. We show the results for $m = 1$, $m = 5$ and $m = 15$. We can see that the deviation θ_t^m for a fixed m roughly increases as the training approaches completion, which has been demonstrated by the gray lines in Figure 2. It means that *if we want to limit the deviation caused by partial aggregation, we have to use a larger m in a later round*. Furthermore, for fixed model accuracy (suggesting the same progress percentage), the non-IID scenario always has a larger deviation under the same m. It means *we should use a larger m to limit the deviation when the data sample distribution is non-IID.*

3.3.3. Summary of observations

Our observations can be summarized as follows and these observations inspire the subsequent method design.

First, partial aggregation is a promising approach because the first some arrived device models in each round have been able to derive a relatively good aggregated model, with only small precision loss. The contribution of arriving device models follows the law of diminishing marginal utility evidently.

Second, how many models shall be waited should be carefully determined. A fixed value may not be suitable. At least two factors must be taken into account, *i.e.*, non-IID ratio and training progress percentage. In order to keep the deviation under an expected value in each single round, the server needs to wait for more device models when non-IID ratio is higher or when the training task approaches completion.

4. Implementation of Partial Aggregation Federated Learning

The experiments described in Section 3 demonstrate that partial aggregation can reduce the training time of a single round without losing much precision, which indicates that partial aggregation is a promising approach to accelerate the whole training process. At the same time, the experiments also show that the aggregation number of each round (*i.e.,* m_t) should be a key to the performance of partial aggregation. We name this issue as *waiting strategy*. The other key issue is that how to aggregate all received device models to obtain the global model for the next round (*i.e.*, $\mathcal{A}$), which is named as *aggregation strategy*.

Let us assume that a device k receives the global model aggregated in round $t-1$ (*i.e.*, w_{t-1}). It conducts local training starting from w_{t-1} and gets an updated device model. This device model is expected to be used in the aggregation of round t, therefore it is regarded as a device model of round t. Assume that the server is waiting for device models to conduct aggregation to obtain the global model of round t' when a device model of round t arrives. If $t' = t$, the device model is called as a *fresh model*. Otherwise, we have $t' > t$, which means the device model arrives later than expected, and it is called *stale model* and $t' - t$ is used to measure the *staleness* of the device model.

A stale model arrives later than expected, but obviously it may have valuable information to improve the precision of the global model in the round, especially when the local models from this device have never been aggregated in global models of earlier rounds. Furthermore, if only fresh models are considered in the aggregation, the obtained global models tend to overfit the data samples in the devices that are always fast, and the samples in slow devices cannot make contribution to the global models, which degrades the training accuracy. Therefore, stale models must be considered during aggregation. On the other hand, since stale models are trained from expired global models, placing too much emphasis on stale models can have negative influence on the convergence of the training process. If the staleness exceeds the tolerance range predefined, the device model should not participate in aggregation. In other words, stale models provide corrections to the optimization direction of the global model, but they also drag down the training progress. How to efficiently use stale models is the key issue in designing the aggregation strategy.

In summary, there are two key problems in our partial aggregation federated learning system as follows.

- *waiting strategy*, *i.e.*, How to determine the aggregation number of a round?
- *aggregation strategy*, *i.e.*, How to aggregate the collected client models, either fresh or stale, to derive the global model in a round?

In this section, we will describe how we solve the above two problems and implement our federated learning system.

4.1. Waiting strategy: determining aggregation number via reinforcement learning

Our partially-aggregation federated learning is in fact a Markov Decision Process (MDP). In each round, the server

performs an action to determine the aggregation number of this round. Then the server derives a global model by aggregating received device models and this process changes the state of the training progress. The goal is to help the server find an appropriate action to minimize the total training time to reach the required model accuracy.

4.1.1. DRL (Deep Reinforcement Learning)

Reinforcement learning (RL) is a well-known popular method to analyze problems related to MDP [35]. RL aims to enable an agent to take the best action according to the current state to maximize its long-term gains. The recent success of RL [36, 37, 38] shows that RL agent can learn from the interaction with a complex and dynamic environment. The interaction between the agent's action and state through the environment is denoted by a sequence of (*state, action, reward, new state*). At each time step t, the agent observes the current state s_t and chooses the action a_t accordingly. Its action interacts with the environment and transforms the system into a new state s_{t+1}. By transforming to a new state, the system might be more close to the goal, which means the action produces a reward, denoted by r_t. The reward of action can be negative. By obtaining and analyzing a series of traces (s_t, a_t, r_t, s_{t+1}), a RL agent can learn the best action when facing an any state to maximize long-term gains, *i.e.*, the cumulative discounted reward $R = \sum_{t=1}^{T} \gamma^{t-1} r_t$, where γ is a factor discounting future reward and T is the total number of state transitions.

In the traditional value-based RL algorithm, the agent maintains a $Q(s_t, a_t)$ value table, which stores an estimation of the expected cumulative reward achieved by performing the action a_t at the state s_t. Assuming the table has been known, at each state s_t, the agent searches the Q value table and finds the best action $a_t = \max_a Q(s_t, a)$.

When the state space is continuous or large, it is impractical to store all the information by using Q value table alone, and deep neural network (DNN) can be used as an approximation for the Q value table, which is referred to as *Deep Reinforcement Learning* (DRL). DRL trains a deep neural network ϕ to predict Q values by calculating the function $Q(s_t, a; \phi)$, and ϕ should be able to minimize the MSE loss of prediction $l(\phi)$. Mathematically, the MSE loss of a network ϕ is defined as follows.

$$l(\phi) = (r_t + \gamma \max_a Q(s_{t+1}, a; \phi) - Q(s_t, a_t; \phi))^2 \qquad (2)$$

In this work, we exploit deep Q-network (DQN) [39] to determine the optimal aggregation number m_t in each round of a federated training task. When applying the DQN technique to our problem, we have to define the state, action and reward properly and get a series of (s_t, a_t, r_t, s_{t+1}) for the training of ϕ for our problem. In the following subsections, we explain how we define state, action and reward to solve the problem.

4.1.2. State

"State" should be able to describe the state of a federated learning system clearly. Especially it must include all factors that have an influence on the decision of the optimal aggregation numbers. The experiments in Section 3 reveal two factors: non-IID ratio, training progress percentage (can be reflected by the accuracy of current global model). Besides the above two factors, we conjecture that capability heterogeneity of devices is also an important factor. Devices may have widely varied capability of computation and communication, which makes the time to receive a device model from different devices vary a lot. The distribution of the time may affect the expected time taken to wait for one more device model.

Therefore, a state s_t can be represented by a vector $[\chi^t, P^t, d^t]$. χ^t is the accuracy of current global model before training in this round, which characterizes the training progress percentage. P^t characterizes non-IID data distribution. d^t characterizes capability heterogeneity of devices.

P^t is a vector and each element in the vector P (denoted by P_i^t) is defined as the standard deviation of the $i-$th element in vector $p_k (k \in C_t)$. The $i-$th element in vector p_k (denoted by $p_{i,k}$) is the proportion of data samples of the $i-$th label to all data samples of this device. Mathematically, we define $p_{i,k} = \frac{N_{i,k}}{N_k}$, where N_k is the number of data samples on the device k and $N_{i,k}$ is the number of data samples with the $i-$th label on the device k. Please note $\sum_i p_{i,k} = 1$ holds for any k.

Here, we assume that the server can learn $p_{i,k}$ from devices. It is not necessary for devices to report their data samples directly, and we think $p_{i,k}$ does not expose the privacy of devices. If a device is very sensitive to privacy, it can add a certain amount of noise when reporting its $p_{i,k}$.

d^t is defined as the standard deviation of single-round training time of devices selected in the round t. The single-round training time of a particular device can be estimated from its communication latency of a device model between the device and the server and its computation time taken to train a device model using all its data samples. The communication latency is impacted by the size of the model and the available bandwidth, while the computation time is impacted by the number of data samples, the model, and CPU or GPU frequency. An estimation of communication latency and computation time has been proposed in [40].

4.1.3. Action

The action of an agent in this problem is to select a particular aggregation number m_t. In each round t, a DQN agent calculates $Q(s_t, a; \phi)$ for each action a (*i.e.*, a particular aggregation number) in the current state s_t using the forward propagation calculation in neural network of agent. Then the agent selects the action with the greatest expected reward, *i.e.*, $\max_a Q(s_t, a; \phi)$. For example,

assume there are N clients in total and the server selects a subset C_t of clients to participant in each round, the RL agent will output the values of $Q(s_t, a; \phi)$ for all $|C_t|$ actions, *i.e.*, aggregation number from 1 to $|C_t|$.

This naive design binds the neural network structure with the number of selected participants in a round and it is not flexible when the number of selected participants changes. To make the agent more flexible, our RL agent will always output 10 values and each value is the Q value of an aggregation percentage instead of an aggregation number. In this way, our RL agent can still work when the number of participants changes.

4.1.4. Reward

We always prefer to see a more accurate global model and shorter single-round training time. In DRL, the definition of reward function should be able to reflect the preference for a state. Therefore, there should be two factors in the reward function, *i.e.*, accuracy of current global model in the new state, and the time consumption to conduct this round of training. We define the reward value from taking action a_t in state s_t as follows:

$$r_t = \beta^{\chi^{t+1}-\chi^*} - \tau^t \tag{3}$$

where χ^{t+1} is the accuracy of the global model after training in this round, χ^* is the target model accuracy that the learning task expects to reach, β is a base number of the exponential item which amplifies the reward for higher model accuracy, τ^t is the normalized time cost in this round.

In Equation 3, $\beta^{\chi^{t+1}-\chi^*}$ is to reflect the preference for global model accuracy and we should have $\beta > 1$. It is because the expected increment of model accuracy from a single round generally decreases as the model accuracy becomes higher, *i.e.*, it is more difficult to improve the accuracy of the global model in later rounds. Therefore, the reward for the same increment of model accuracy should be larger as χ^t increases.

τ^t is to reflect the preference for a smaller single-round training time. Particularly, we define $\tau^t = \frac{\mathbb{T}^t}{B}$, in which $\mathbb{T}^t$ is the training time of round t, and B is set to be the training time of the first device model received by the server in the first round. We can assume that B is fixed in all rounds for a particular training task. By normalizing $\mathbb{T}^t$ by B, we try to avoid putting too much more weight on single-round training time than model accuracy, especially in complex training tasks that are expected to have a very large τ^t.

Through the reward function, the reinforcement learning agent evaluates the performance of the current aggregation number, including the improvement of the model accuracy and the time cost of the current aggregation number. Thus the RL agent can learn and predict the performance reward of each aggregation number. In other words, the RL agent implicitly learns the marginal performance increase brought by one more client model and determines whether to wait for the next model.

4.1.5. Training the waiting strategy

We use double deep Q-learning network (DDQN) [41] to train the waiting strategy. Training a DQN directly might be unstable since it evaluates the estimation $Q(s_t, a_t; \phi)$ and compares it with $r_t + \gamma \max_a Q(s_{t+1}, a; \phi)$ instead of the real optimal Q value when computing the loss. Thus, the comparison benchmark varies with ϕ, which causes the learning process unstable and slow. DDQN uses two DQNs. One DQN ϕ is updated and used for decision in a single time step. The other DQN $\phi^{'}$ is used to evaluate $Q(s_t, a_t; \phi)$ and it is updated every M rounds, which reduces jitter during the Q evaluation and learning process.

To train the DRL agent, the server first selects clients randomly and initializes the federated learning model to initialize the state. The DQN ϕ takes the state and generates an action to decide aggregation number, which has the greatest Q value estimation. In each round, the pair of (s_t, a_t, r_t, s_{t+1}) is collected. After several rounds, the DRL agent has sampled a few action-state pairs, with which the agent can learn to minimize Equation 2 as:

$$l_t(\phi_t) = (Y_t - Q(s_t, a_t; \phi_t))^2 \tag{4}$$

where Y_t is the target of comparison in round t and

$$Y_t = r_t + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \phi); \phi^{'})$$

The ϕ is updated to minimize the loss by gradient descent as follows.

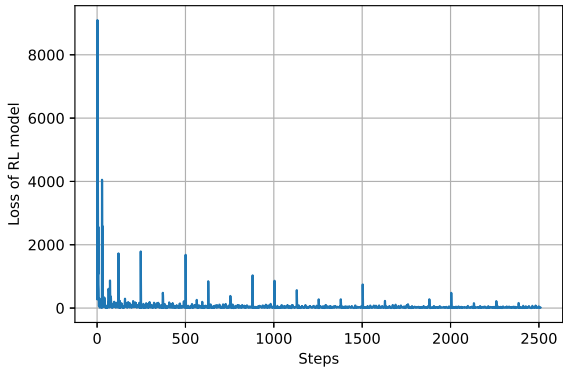
$$\phi_{t+1} = \phi_t + \hat{\eta}(Y_t - Q(s_t, a_t; \phi_t)) \bigtriangledown_{\phi_t} Q(s_t, a_t; \phi_t)$$

where $\hat{\eta}$ is the learning rate of the DRL agent.

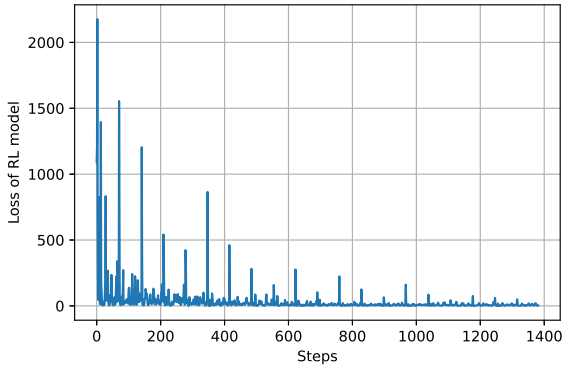
The RL agent is trained for several episodes before applying to a new training task. In this paper, we use a two-layer MLP [42] network as the model for the DL agent and the DRL agent for each task is trained for 20 episodes. The convergence curve of the RL model is shown in Figure 4. Please note that an episode includes many rounds, and the RL agent learns the reward of an aggregation number every round. Figure 4 shows the value of loss function (Equation 2) of the RL model in each round. Experiment results suggest that the MLP network can converge and the RL agent can learn the reward of a aggregation number effectively. The trained RL agent will be applied to predict the aggregation number in experiments in Section 5 later.

4.2. Aggregation strategy: aggregating fresh and stale models

In federated learning, the server keeps receiving client models from devices, and it has to aggregate these local models properly to derive a global model. Currently, there are two kinds of aggregation methods. One is average method [9], in which the global model is a result of taking average on all received models. Mathematically,



(a) DRL training on CIFAR-10



(b) DRL training on MINIST

Figure 4: Training the DRL agent

Aggregation in average manner:

$$\mathcal{A}_{avg}(S): \quad w_t = \sum_{k \in S} \frac{|D_k|}{|\bar{D}|} w_t^k$$

where S represents the set of device models to be aggregated and $\bar{D}$ is the set of data samples in S totally. Aggregation in average manner helps mitigate the impact of outlier device models that deviate from the majority and it considers the contribution of each model equally.

The other method is a kind of weighted method [11]. This method aggregates local models one by one. Once a local model is received, the server combines the current global model with the local model in a weighted manner. Mathematically,

Aggregation in weighted manner:

$$\mathcal{A}_{weight}(w, \alpha): w_t = (1-\alpha) w_{t-1} + \alpha w$$

where w represents the model newly received and α is a weighting factor between the new device model and the current global model derived from device models received before. In fact, the above equation is an exponentially weighted moving average algorithm, which can be expanded as follows.

$$w_t = \alpha(w + \sum_{k=0}^{t-1} (1-\alpha)^{t-k} w_k) \tag{5}$$

The expanded equation clearly shows that the method weights later device models more than earlier device models. This design is based on an assumption that later models are likely to be more accurate than earlier device models. This assumption holds when later models are derived from more accurate global models (later global models), but it does not hold if the later model is derived from an earlier global model and it arrives later just because the device is too slow. If α is too large, the global model might be very sensitive to new models and it is likely that new models from slow devices drag back the training progress.

4.2.1. Design of our aggregation strategy

In this work, the device models received by the server in a round can be fresh or stale. Fresh models are based on the latest global model and are likely to be more accurate, while stale models are based on earlier global models and they are derived from data samples of slow devices. Obviously, fresh device models and stale device models should not be treated equally.

Therefore, we divide the received models into two sets: the set of fresh models S_t and stale models $\hat{S}_t$. Within one set, we use the average method to mitigate the impact of abnormal models that deviate from the majority.

$$w'_t = \mathcal{A}_{avg}(S_t) \tag{6}$$

$$w''_t = \mathcal{A}_{avg}(\hat{S}_t) \tag{7}$$

where w'_t represents the aggregation model of fresh models and w''_t represents the aggregation model of stale models.

Then we use the weighted method to combine w'_t and w''_t, wherein w'_t represents fresh models that tend to be more accurate and w''_t represents stale models that might be less accurate but have valuable information about data samples in slow devices.

$$w_t = (1-\alpha_t) w'_t + \alpha_t w''_t \tag{8}$$

The key issue here is how to set α_t properly. Intuitively, when the stale models are less stale (smaller staleness) or the stale models represent a lot of data samples, α_t should be larger. We have

$$\alpha_t = \frac{|D_t^{''}|}{|D_{t|}^{'}| + |D_t^{''}|} e^{\tau_t} \tag{9}$$

where τ_t is the average staleness in $\hat{S}_t$, $|D_t^{'}|$ is the the set of data samples in S_t and $|D_t^{''}|$ is the set of data samples in $\hat{S}_t$.

Please note that α is dynamic to be adaptive to the received models in a single round, *i.e.*, α_t is different for different t and it is not a fixed value. In the next subsection, we conduct experiments to illustrate the necessity and benefit of a dynamic weighting factor.

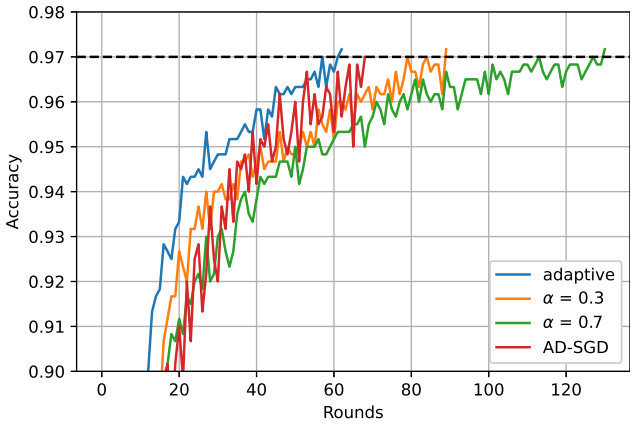
4.2.2. Necessity for the dynamic α_t

We conduct an experiment to compare the results of a changing α_t (according to Equation 9) and two fixed setting ($\alpha = 0.3$ in all rounds and $\alpha = 0.7$ in all rounds). We also compare our results with the AD-SGD algorithm proposed by Li et.al [32] since our method and AD-SGD both attempt to mitigate the negative impact of stale models in utilizing them. AD-SGD algorithm tries to aggregate normal and stale models and uses Hessian approximation matrix to mitigate the negative impact of stale models. This algorithm requires clients to transmit model parameters and gradient parameters to the server in each round for Hessian approximation computation, while our method only requires model parameters and has a lower communication overhead.

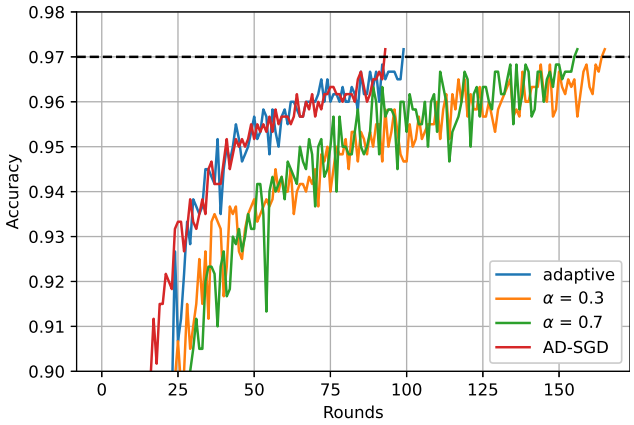
In the experiment, we use TensorFlow to train a two-layer CNN model on the MINIST dataset. The dataset contains a training set with 60,000 data samples, and a test set of 10,000 data samples. There are 100 devices in total and each device is simulated by an individual thread. The CNN model is trained round by round until it reaches an accuracy of 97% in the test set. In each round t, the server randomly selects 10 clients to participate, *i.e.*, $|C_t| = 10$. To focus on the impact of aggregation strategy, we fix the aggregation number as half of the participants in each round in this experiment, *i.e.* $m_t = 5$ in all rounds.

We conjecture that the non-IID ratio may have an influence on the comparison between different aggregation strategies. When the non-IID ratio is larger, each device is likely to be unique and its data samples must be included in the global model, therefore stale models might be more valuable. Therefore we conduct experiments for two cases: IID and non-IID. The setting of non-IID is the same as the experiment in Section 3.

We plot the model accuracy in every round during the training in Figure 5. From it, we can see that our dynamic α_t setting can bring faster convergence rate than the fixed settings in both cases. It demonstrates that the dynamic α_t strikes a balance between the values of stale models and the negative effect of model staleness. Our method achieves a convergence speed similar to the AD-SGD algorithm but the required data transmission volume between clients and the server in our method is only half of that in AS-SGD. In mobile edge scenarios, bandwidth resources are scarce and data transmission is time-consuming. Transmitting less data means saving more transmission time and thus reducing the total training time.



(a) IID



(b) non-IID

Figure 5: Model accuracy in every round during the training

4.3. The learning system

Based on our waiting strategy and aggregation strategy, we design a learning system which is shown in Figure 6.

Our learning system contains three blocks: interfaces to devices, RL agent and partial model aggregator. The interface block consists three modules that are responsible for three tasks. The information collector stores the information about devices, such as the distribution of data samples among devices and the computation and communication capability of each device. The client selector and global model distributor module is responsible for selecting participants from all available clients in each round and distribute current global model to the selected participants. The client model receiver module is responsible for collecting client models from participants. The RL agent block determines the aggregation number in each round using our waiting strategy. The partial model aggregator aggregates fresh and stale client models to obtain a global model in each round.

With our learning system, in each round, the client selector randomly selects a subset of clients to participate and the selector informs the reinforcement learning agent of the selected clients in step 1. The reinforcement learning agent fetches information about the data distribution, capabilities of the selected clients and training progress from the information collector in step 2. In step 3, the RL agent decides the aggregation number and informs the partial model aggregator.

Meanwhile, the global model distributor sends the current global model to the selected clients, which is listed as

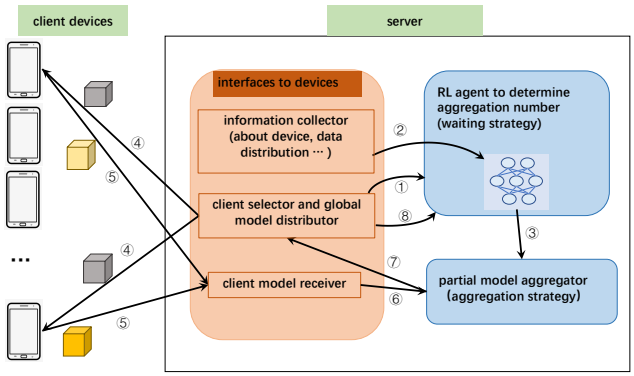


Figure 6: System overview

step 4 but in fact can start once step 1 is finished. The client model receiver keeps receiving the models trained locally by clients and forward them to the partial aggregator, which is shown as the step 5 and step 6. After the number of client models received by the partial model aggregator reaches the aggregation number determined by the RL agent, the aggregator aggregates both fresh and stale models to obtain the global model of this round in step 7. The global model is evaluated on the validation set and the accuracy is measured. The training time of this round and the accuracy of the global model is sent to the RL agent to calculate the reward for the learning of our waiting strategy in step 8. This completes a single round. The system runs round by round until the training task is completed.

5. Experiment

We implement FedPA using TensorFlow and conduct several simulation experiments to evaluate the performance of FedPA.

We use a two-layer MLP [42] network with 64 hidden states as the model for the DL agent in FedPA. We choose this simple MLP network because it has fewer parameters to be learned and converges more easily than models with a complex neural network. Although a complex model may achieve better performance, its design needs careful considerations and it is not the focus of this work. In our FedPA, the DRL agent is trained for 20 episodes and the MLP network has converged. Then it is applied in FedPA and we start to collect statistics of the experiments for our performance evaluation.

The simulation experiments are run on a server. In our experiments, we simulate 100 user devices. Each user device is simulated by a thread and it occupies a CPU exclusively. In each round, the server randomly selects 10 clients to participate, *i.e.*, $|C_t| = 10$, which means 10 threads are wakened up to train device models in each round.

We compare our FedPA with three algorithms, *i.e.*, FedAvg, FedAsync [11] and FLANP [31].

- FedAvg. In each round, the server waits for device models until all device models are received or a preset timer expires. It averages them to get an updated global model. In order to make sure the algorithms are comparable, FedAvg algorithm is also set to randomly select 10 clients to participate in each round.

- FedAsync. The server periodically sends the current global model to a client randomly. The server aggregates device models using a weighted average method every time it receives a model.

- FLANP. FLANP sorts client nodes from fast to slow according to their completion time. It uses first several fast nodes for warm-up training, and gradually involves more participants to train local models in the training process until the model reaches a satisfactory accuracy. It always chooses client nodes from fast to slow in each round and waits for all selected clients to return results before entering the next round. Thus, FLANP has no stale models. In the FLANP algorithm, when to involve more number of participants depends on the value of loss function of the current model and a threshold hyper-parameter. In practice, the threshold hyper-parameter is difficult to determine or requires prior knowledge to determine. To ensure fairness, we tune its threshold hyper-parameter to ensure that the summations of the number of selected participants in each round of these two algorithms are close, which means two algorithms consume approximately equal computational resources. We compare the performance of FLANP with ours under different non-IID ratios.

- FedPA-fixed. We also simulate FedPA with a fixed aggregation number, in which there is no reinforcement learning agent to learn dynamic aggregation numbers. We refer it as *FedPA-fixed*.

The training tasks are based on two data sets: CIFAR-10 and MNIST. We run each algorithm to train a CNN model using each data set and compare the accuracy of the models achieved by these algorithms. We choose the hyper-parameter with the best performance for FedAvg, and the hyper-parameter is different on different data sets.

CIFAR-10: CIFAR-10 dataset is used widely in Federated learning studies. It is an object classification dataset, which consists of 50,000 training images and 10,000 testing images with 10 object classes. The training task is to train a CNN model with two 5×5 convolution layers using this dataset. Both layers have 64 channels, and each convolution layer is followed by a pooling layer and a local response normalization layer, and the batch size is set to be 32.

MINIST: MINIST is a classic handwriting image classification data set, which consists of 60,000 training images and 10,000 testing images with 10 classes (digit 0 to 9). The training task on this dataset is to train a CNN model

with two 5×5 convolution layers. The first layer has 32 channels and the second layer has 64 channels. Each convolution layer is followed by a pooling layer, and the batch size is set to be 50.

The target accuracy for the CNN training on the MNIST dataset is 96%, and the target accuracy is 55% for the CNN training on CIFAR-10. Previous works have shown that the two-layer CNN model converges when the model almost reaches the corresponding target accuracy in the corresponding datasets with FedAvg method.

There are two important factors that affect the performance of these algorithms, *i.e.*, the distribution of data samples among devices (characterized by non-IID ratio) and the levels of device capability heterogeneity. Our experiments try to evaluate the performance in various situations.

5.1. Performance comparison under different non-IID ratios

In this subsection, we try to evaluate the performance when the non-IID ratio changes. We use the same approach to generate non-IID data samples as introduced in Section 3. We conduct experiments to explore the performance of various algorithms under three settings, IID, non-IID with $\sigma = 0.5$ and non-IID with $\sigma = 0.8$.

We plot the testset accuracy during training in Figure 7. We plot a horizontal line to emphasize the time taken by each algorithm to reach the target accuracy. We can see that time taken by FedPA is shorter than other algorithms in most cases. Our partial model aggregation strategy converges in all cases and avoids the disadvantage of FedAsync.

In the experiment using the CIFAR-10 dataset, when the data distribution is IID or the non-IID ratio is small ($\sigma = 0.5$), both FedPA and FedAsync outperform FedAvg, which demonstrates the benefit of discarding the requirement of being synchronous, and FLANP outperforms FedAvg since it prefers to select fastest nodes during training. The performance of FLANP and FedPA are close when non-IID ratio is small ($\sigma = 0.5$), but the performance of FLANP oscillates slightly and has spikes. The reason for the spikes is as follows. When the distribution of data samples among participants is non-IID and the global model overfits easily if the server always prefers a small number of fast participants. FLANP always prefers those fast nodes during training, therefore the difference among participants causes that local trained models have relatively large deviation from the optimal global model, and makes the performance of the global model oscillates when the number of participants changes. When the non-IID ratio σ reaches 0.8, the convergence rate of FedAsync significantly slows down and the model cannot achieve the target accuracy. FLANP oscillates more often and takes more time to finish the training task. In the experiment on the MINIST dataset, the FedAsync strategy converges slowly and cannot achieve the target accuracy in all IID and non-IID settings. FedPA takes less training time than FLANP in non-IID settings.

In summary, our experimental results show that our strategies take less training time than the FedAvg and the FedAsync method in both IID and non-IID scenarios. Our strategies also take less training time than the FLANP algorithm in non-IID scenarios.

5.2. Performance comparison under different levels of device capability heterogeneity

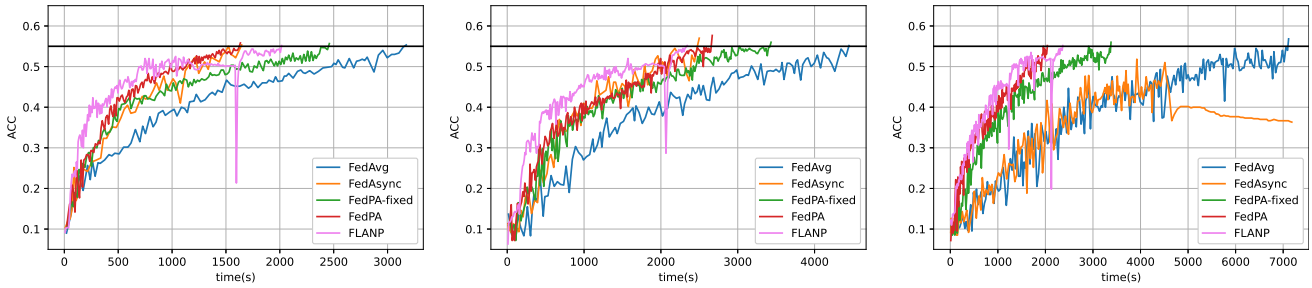
The time a device takes to train a device model in a round depends on the computation capability of the device, and the time for a device to communicate with the server to retrieve the global model and submit its device model depends on the bandwidth between the device and the server. Devices that participate in a training task may have different computation capabilities and communication capabilities.

Each device is simulated by a thread running on a CPU core exclusively. Because all threads are running on the same type of CPU cores, the running time, *i.e.*, the time length that the CPU takes to finish a single-round training, is approximately the same. We let the threads sleep for a while before uploading device models, and devices with different capabilities are simulated by setting the ratio of sleep time to running time to be different for devices. The ratio is retrieved by sampling points on a truncated normal distribution. By controlling the deviation parameter (denoted as h) of the truncated normal distribution, we can have different levels of device capability heterogeneity.

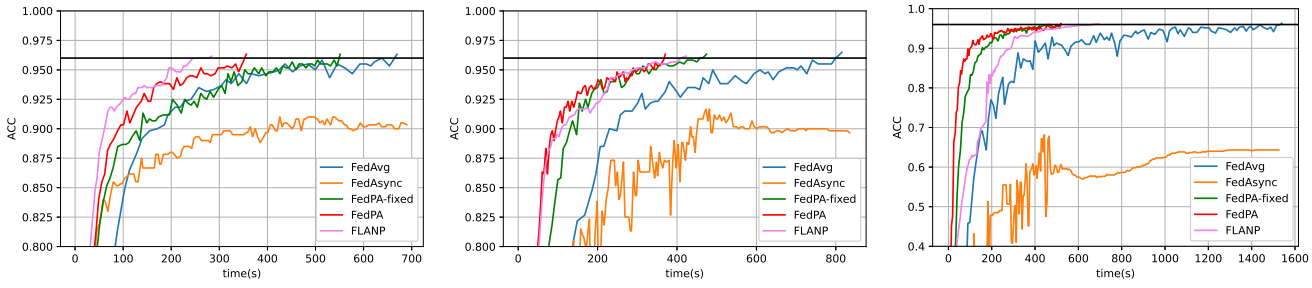
In this subsection, we try to evaluate the performance in two kinds of distribution of device capability, *i.e.*, $h = 1$ and $h = 5$, which are shown in Figure 8. In the distribution of $h = 1$, most devices are powerful and they can complete their jobs fast, but there are a small number of devices that are extremely slow. In the distribution of $h = 5$, the ratios are widely distributed in an interval without any long tail, which means the devices have less significant capability difference compared to $h = 1$. Both distributions are normalized, which means two settings have an equal expected average capability of devices.

Figure 9 shows the testset accuracy during the two training tasks in two different scenarios, *i.e.*, $h = 1$ and $h = 5$. The time taken by FedPA is shorter than FedAvg and FedAsync in all cases. The time taken by FedPA is close to the time taken by FedPA-fixed in three cases. In the case on CIFAR-10 and $h = 1$, FedPA clearly outperforms FedPA-fixed, which shows that the dynamic aggregation number is especially helpful when devices are more heterogeneous.

In the two cases on the MINIST dataset, FedAsync cannot achieve the target accuracies. It only achieves an accuracy of 67% in the case of $h = 1$ and achieves an accuracy of 75% when $h = 5$. We believe it demonstrates the negative effect of starting the next training round by only waiting for one client model. As illustrated in Figure 2 in Section 3, considering only one client model can



(a) IID (b) non-IID: $\sigma = 0.5$ (c) non-IID: $\sigma = 0.8$



(d) IID (e) non-IID: $\sigma = 0.5$ (f) non-IID: $\sigma = 0.8$

Figure 7: Accuracy during training under different non-IID ratios (top: CIFAR-10, bottom: MINIST)

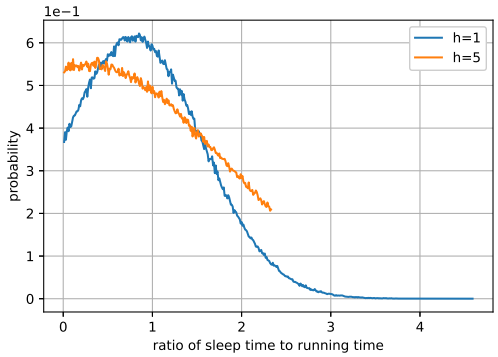


Figure 8: density (probability) v.s. ratio of sleep time to running time

bring large deviation compared with the expected global model. In our solution, we consider multiple client models and the deviation quickly decreases before aggregation. It successfully avoids the problem of FedAsync mentioned above.

5.3. The output of RL agent under different non-IID ratio

In this subsection, we present the prediction results of our RL agent in scenarios with different non-IID ratios to explore how the RL agent adapts to different non-IID ratios.

We conduct experiments for scenarios with different non-IID ratios (*i.e.*, IID, non-IID with $\sigma = 0.5$ and $\sigma = 0.8$) and different distribution of device capability (*i.e.*, $h = 1$ and $h = 5$). The experiment results are presented in Table 1 and 2. We can see that the RL agent predicts a larger aggregation number when the non-IID ratio is larger and predicts a smaller aggregation number when the heterogeneity of device capability is larger.

Table 1: The average of predicted aggregation number on CIFAR-10

	IID	**non-IID:** $\sigma = 0.5$	**non-IID:** $\sigma = 0.8$
$h = 1$	4.237	5.143	5.304
$h = 5$	3.851	5.090	5.120

Table 2: The average of predicted aggregation number on MINIST

	IID	**non-IID:** $\sigma = 0.5$	**non-IID:** $\sigma = 0.8$
$h = 1$	3.143	3.703	5.229
$h = 5$	2.837	3.621	4.093

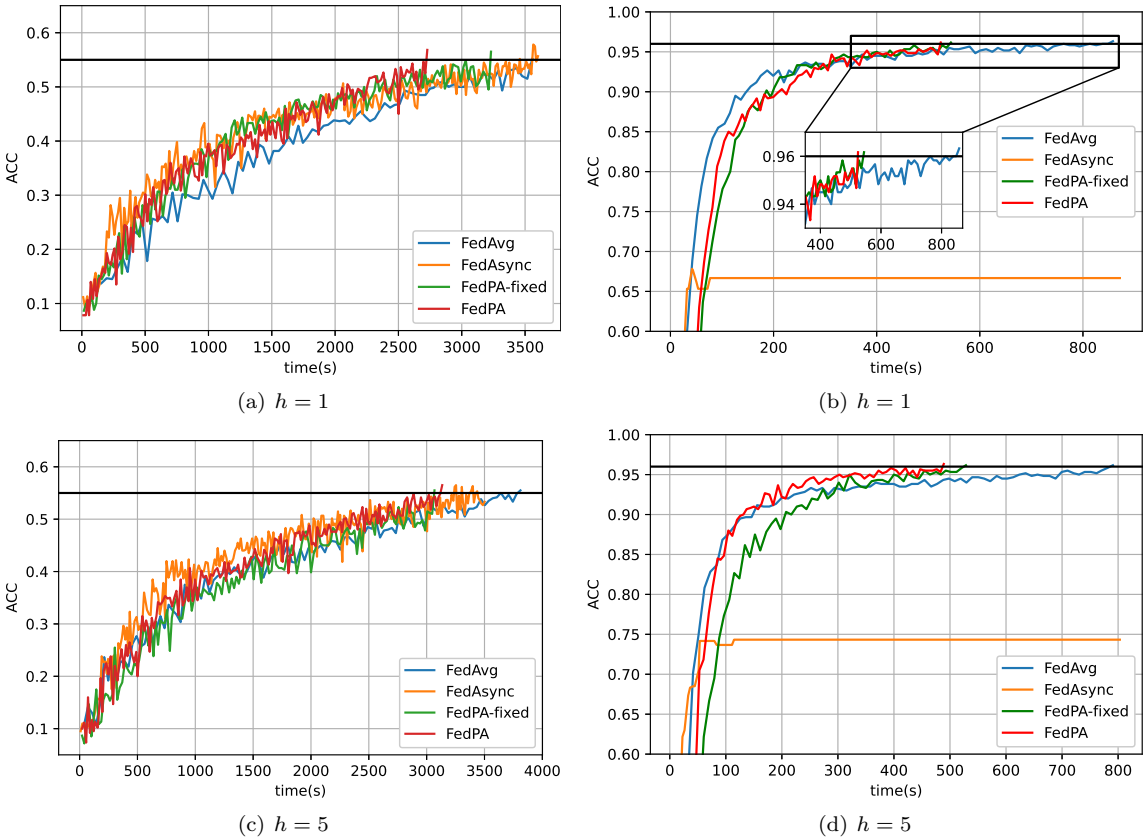
5.4. Privacy and robustness of our RL agent

Privacy protection is an advantage of federated learning. In our method, the input of RL agent requires the clients to send the server their data distribution information, which does not expose raw data of clients. If the clients are extremely sensitive to privacy, they can add a certain amount of noise to their data distribution information, as people do in differential privacy. In this subsection, we explore the robustness of our RL agent when noise is added to the percentage of each individual class label.

We add Laplace noise to the percentages and the probability density function of noise is as follow.

$$f(x) = \frac{1}{2\lambda} e^{\frac{-|x-\mu|}{\lambda}} \tag{10}$$

where x is the percentage of each individual class label, μ is the average of the noise and we set it to 0 in this



(a) $h = 1$ (b) $h = 1$

(c) $h = 5$ (d) $h = 5$

Figure 9: Accuracy v.s. time on different level of heterogeneity (left: CIFAR-10, right: MINIST)

Table 3: Training performance with different scales of noise on CIFAR-10

scale	avg. aggregation number	num. of iterations	duration
0	5.143	180	2668.33
1	5.013	155	2641.95
3	5.198	177	2707.83
5	4.989	190	3213.52

Table 4: Training performance with different scales of noise on MINIST

scale	avg. aggregation number	num. of iterations	duration
0	3.703	77	370.49
1	4.896	68	382.55
3	5.329	71	408.19
5	4.867	76	483.30

experiment, λ is the scale of the noise. We compare the average of predicted aggregation number, the number of iterations and the training duration under different scales of noise.

Table 3 and 4 show the experimental results on dataset CIFAR-10 and MINIST. The predicted aggregation number by RL agent and the number of iterations are stable on dataset CIFAR-10. These two metrics are also stable on dataset MINIST when the scale is no more than 1. The training duration is stable on both datasets when the scale is less than 5. In summary, our RL agent is robust within a certain amount of noise.

6. Conclusion

Federated learning is a promising approach to train models using the large amounts of data and the AI components owned by end devices. However, end devices are heterogenous and unstable. They are not dedicated to train models and their capability and availability cannot be guaranteed during training. Federated Averaging, the most widely accepted framework, suffers seriously from participant devices with weak computation and/or communication capability.

In this paper, we present the concept of partial model aggregation and conduct experiments to show that the first several arrived device models in each round have been able to derive a relatively good aggregated model with only small precision loss. The deviation of partial aggregation model is large at the beginning, but decreases quickly with more client models collected. The experiments also suggest that the deviation of partial aggregation model for a specific aggregation number is influenced by the non-IID ratio and training progress percentage. We further

propose two strategies, waiting strategy and aggregation strategy, to solve the two key issues in our partial aggregation framework. Particularly, our waiting strategy determines the aggregation number for each round via reinforcement learning. It adapts to various scenarios very well and makes sure that the server only waits for the device models with significant contribution. Our aggregation strategy classifies fresh models and stale models, and gives appropriate weights to them. Stale models have valuable information to improve the precision of the global model in the round, but they are trained from expired global models which means they can have a negative influence on the convergence of the training process. Our weighting factor takes model staleness and the number of data samples used by stale models into consideration. The experiments demonstrate that FedPA performs better than FedAvg and other three algorithms named FedAsync, FLANP and AD-SGD, especially when the devices are more heterogeneous in terms of the data and capability owned by them. The experiments also suggest that FedPA is robust when a certain amount of noise is added into the input for privacy concerns.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant 62072269 and in part by the Natural Science Foundation of China under Grant 61772139.

References

[1] J. Kooistra, Newzoo's 2018 global mobile market report: Insights into the world's 3 billion smartphone users, Newzoo, September 11 (2018).

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

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[4] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal processing magazine 29 (6) (2012) 82–97.

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

[7] R. Gu, S. Yang, F. Wu, Distributed machine learning on mobile devices: A survey, arXiv preprint arXiv:1909.08329 (2019).

[8] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, X. Chen, Convergence of edge computing and deep learning: A comprehensive survey, IEEE Communications Surveys & Tutorials 22 (2) (2020) 869–904.

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

[10] 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).

[11] C. Xie, S. Koyejo, I. Gupta, Asynchronous federated optimization, arXiv preprint arXiv:1903.03934 (2019).

[12] W. Wu, L. He, W. Lin, R. Mao, C. Maple, S. A. Jarvis, Safa: a semi-asynchronous protocol for fast federated learning with low overhead, IEEE Transactions on Computers (2020).

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

[14] M. Duan, D. Liu, X. Chen, Y. Tan, J. Ren, L. Qiao, L. Liang, Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications, in: 2019 IEEE 37th International Conference on Computer Design (ICCD), IEEE, 2019, pp. 246–254.

[15] H. Eichner, T. Koren, H. B. McMahan, N. Srebro, K. Talwar, Semi-cyclic stochastic gradient descent, arXiv preprint arXiv:1904.10120 (2019).

[16] K. Hsieh, A. Phanishayee, O. Mutlu, P. B. Gibbons, The non-iid data quagmire of decentralized machine learning, arXiv preprint arXiv:1910.00189 (2019).

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

[18] J. Wu, W. Huang, J. Huang, T. Zhang, Error compensated quantized sgd and its applications to large-scale distributed optimization, arXiv preprint arXiv:1806.08054 (2018).

[19] Y. Lin, S. Han, H. Mao, Y. Wang, W. J. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training, arXiv preprint arXiv:1712.01887 (2017).

[20] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, K. Gopalakrishnan, Adacomp: Adaptive residual gradient compression for data-parallel distributed training, arXiv preprint arXiv:1712.02679 (2017).

[21] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns, in: Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[22] Y. Chen, X. Sun, Y. Jin, Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation, IEEE Transactions on Neural Networks and Learning Systems (2019).

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

[24] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, O. Mutlu, Gaia: Geo-distributed machine learning approaching {LAN} speeds, in: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), 2017, pp. 629–647.

[25] L. G. Valiant, A bridging model for parallel computation, Communications of the ACM 33 (8) (1990) 103–111.

[26] B. Recht, C. Re, S. Wright, F. Niu, Hogwild: A lock-free approach to parallelizing stochastic gradient descent, in: Advances in neural information processing systems, 2011, pp. 693–701.

[27] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, E. P. Xing, More effective distributed ml via a stale synchronous parallel parameter server, in: Advances in neural information processing systems, 2013, pp. 1223–1231.

[28] X. Lian, Y. Huang, Y. Li, J. Liu, Asynchronous parallel stochas-

tic gradient for nonconvex optimization, in: Advances in Neural Information Processing Systems, 2015, pp. 2737–2745.
[29] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, E. P. Xing, High-performance distributed ml at scale through parameter server consistency models, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015, pp. 79–87.
[30] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–7.
[31] A. Reisizadeh, I. Tziotis, H. Hassani, A. Mokhtari, R. Pedarsani, Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity, arXiv preprint arXiv:2012.14453 (2020).
[32] X. Li, Z. Qu, B. Tang, Z. Lu, Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients, arXiv preprint arXiv:2102.06329 (2021).
[33] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.
[34] H. Wang, Z. Kaplan, D. Niu, B. Li, Optimizing federated learning on non-iid data with reinforcement learning, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 1698–1707.
[35] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.
[36] H. Mao, M. Alizadeh, I. Menache, S. Kandula, Resource management with deep reinforcement learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 50–56.
[37] H. Wang, D. Niu, B. Li, Distributed machine learning with a serverless architecture, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1288–1296.
[38] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, J. Dean, Device placement optimization with reinforcement learning, arXiv preprint arXiv:1706.04972 (2017).
[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).
[40] N. H. Tran, W. Bao, A. Zomaya, N. M. NH, C. S. Hong, Federated learning over wireless networks: Optimization model design and analysis, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1387–1395.
[41] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, arXiv preprint arXiv:1509.06461 (2015).
[42] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biological cybernetics 59 (4-5) (1988) 291–294.