Optimizing Federated Learning on Non-IID Data with Reinforcement Learning

Hao Wang[1], Zakhary Kaplan[1], Di Niu[2] and Baochun Li[1]
[1]*University of Toronto*, {haowang, bli}@ece.utoronto.ca, zakhary.kaplan@mail.utoronto.ca [2]*University of Alberta*, dniu@ualberta.ca

***Abstract*—The widespread deployment of machine learning applications in ubiquitous environments has sparked interests in exploiting the vast amount of data stored on mobile devices. To preserve data privacy, Federated Learning has been proposed to learn a shared model by performing distributed training locally on participating devices and aggregating the local models into a global one. However, due to the limited network connectivity of mobile devices, it is not practical for federated learning to perform model updates and aggregation on all participating devices in parallel. Besides, data samples across all devices are usually not independent and identically distributed (IID), posing additional challenges to the convergence and speed of federated learning.**

In this paper, we propose FAVOR, an experience-driven control framework that intelligently chooses the client devices to participate in each round of federated learning to counterbalance the bias introduced by non-IID data and to speed up convergence. Through both empirical and mathematical analysis, we observe an implicit connection between the distribution of training data on a device and the model weights trained based on those data, which enables us to profile the data distribution on that device based on its uploaded model weights. We then propose a mechanism based on deep Q-learning that learns to select a subset of devices in each communication round to maximize a reward that encourages the increase of validation accuracy and penalizes the use of more communication rounds. With extensive experiments performed in PyTorch, we show that the number of communication rounds required in federated learning can be reduced by up to 49% on the MNIST dataset, 23% on FashionMNIST, and 42% on CIFAR-10, as compared to the Federated Averaging algorithm.

I. INTRODUCTION

As the primary computing resource for billions of users, mobile devices are constantly generating massive volumes of data, such as photos, voices, and keystrokes, which are of great value for training machine learning models. However, due to privacy concerns, collecting private data from mobile devices to the cloud for centralized model training is not always possible. To efficiently utilize end-user data, Federated Learning (FL) has emerged as a new paradigm of distributed machine learning that orchestrates model training across mobile devices [1]. With federated learning, locally trained models are communicated to a server for aggregation, without collecting any raw data from users. Federated learning has enabled joint model training over privacy-sensitive data in a wide range of applications, including natural language processing, computer vision, and speech recognition.

However, unlike in a server-based environment, distributed machine learning on mobile devices is faced with a few fundamental challenges, such as the limited connectivity of wireless networks, unstable availability of mobile devices, and the non-IID distributions of local datasets which are hard to characterize statistically [1]–[4]. Due to these reasons, it is not as practical to perform machine learning on all participating devices simultaneously in a synchronous or semi-synchronous manner as is in a parameter-server cluster. Therefore, as a common practice in federated learning, only a subset of devices are randomly selected to participate in each round of model training to avoid long-tailed waiting times due to unstable network conditions and straggler devices [1, 5].

Since the completion time of federated learning is dominated by the communication time, Konečný *et al.* [6, 7] proposed structured and sketched updates to decrease the time it takes to complete a communication round. McMahan *et al.* [1] presented the Federated Averaging (FEDAVG) algorithm, which aims to reduce the number communication rounds required by averaging model weights from client devices instead of applying traditional gradient descent updates. FEDAVG has also been deployed in a large-scale system [4] to test its effectiveness.

However, existing federated learning methods have not solved the statistical challenges posed by heterogeneous local datasets. Since different users have different device usage patterns, the data samples and labels located on any individual device may follow a different distribution, which cannot represent the global data distribution. It has been recently pointed out that the performance of federated learning, especially FEDAVG, may significantly degrade in the presence of non-IID data, in terms of the model accuracy and the communication rounds required for convergence [8]–[10]. In particular, FEDAVG randomly selects a subset of devices in each round and averages their local model weights to update the global model. The randomly selected local datasets may not reflect the true data distribution in a global view, which inevitably incurs biases to global model updates. Furthermore, the models locally trained on non-IID data can be significantly different from each other. Aggregating these divergent models can slow down convergence and substantially reduce the model accuracy [8].

In this paper, we present the design and implementation of FAVOR, a control framework that aims to improve the performance of federated learning through intelligent device selection. Based on reinforcement learning, FAVOR aims to

This work is supported by a research contract with Huawei Technologies Co. Ltd. and a NSERC Discovery Research Program.

accelerate and stabilize the federated learning process by learning to actively select the best subset of devices in each communication round that can counterbalance the bias introduced by non-IID data.

Since privacy concerns have ruled out any possibility of accessing the raw data on each device, it is impossible to profile the data distribution on each device directly. However, we observe that there is an implicit connection between the distribution of the training samples on a device and the model weights trained based on those samples. Therefore, our main intuition is to use the local model weights and the shared global model as states to judiciously select devices that may contribute to global model improvement. We propose a reinforcement learning agent based on a Deep Q-Network (DQN), which is trained through a Double DQN for increased efficiency and robustness. We carefully design the reward signal in order to aggressively improve the global model accuracy per round as well as to reduce the total number of communication rounds required. We also propose a practical scheme that compresses model weights to reduce the high dimensionality of the state space, especially for big models, *e.g.*, deep neural network models.

We have implemented FAVOR in a federated learning simulator developed from scratch using PyTorch[1], and evaluated it under a variety of federated learning tasks. Our experimental results on the MNIST, FashionMNIST, and CIFAR-10 datasets have shown that FAVOR can reduce the required number of communication rounds in federated learning by up to 49% on the MNIST, by up to 23% on FashionMNIST, and by up to 42% on CIFAR-10, as compared to the FEDAVG algorithm.

II. Background and Motivation

In this section, we briefly introduce how federated learning works in general and show how non-IID data poses challenges to existing federated learning algorithms. We demonstrate how to properly select client devices at each round to improve the performance of federated learning on non-IID data.

A. Federated Learning

Federated learning (FL) trains a shared global model by iteratively aggregating model updates from multiple client devices, which may have slow and unstable network connections. Initially, eligible client devices first check-in with a remote server. The remote server then proceeds federated learning synchronously in rounds. In each round, the server randomly selects a subset of available client devices to participate in training. The selected devices first download the latest global model from the server, train the model on their local datasets, and report their respective model updates to the server for aggregation.

We formally introduce FL in the context of a C-class classification problem, which is defined over a compact feature space $\mathcal{X}$ and a label space $\mathcal{Y} = [C]$, where $[C] = 1, \cdots, C$. Let $(\boldsymbol{x}, y)$ denote a particular labeled sample. Let $f : \mathcal{X} \to \mathcal{S}$ denote the prediction function, where $\mathcal{S} = \{\boldsymbol{z} | \sum_{i=1}^{C} z_i = 1, z_i \geqslant 0, \forall i \in [C]\}$. That is, the vector valued function f yields a probability vector $\boldsymbol{z}$ for each sample $\boldsymbol{x}$, where f_i predicts the probability that the sample belongs to the ith class.

Let the vector $\boldsymbol{w}$ denote model weights. For classification, the commonly used training loss is cross entropy, defined as

$$\begin{aligned}\ell(\boldsymbol{w}) &:= \mathbb{E}_{\boldsymbol{x},y\sim p}\left[\sum_{i=1}^{C} \mathbb{1}_{y=i} \log f_i(\boldsymbol{x}, \boldsymbol{w})\right] \\ &= \sum_{i=1}^{C} p(y=i) \mathbb{E}_{\boldsymbol{x}|y=i}[\log f_i(\boldsymbol{x}, \boldsymbol{w})],\end{aligned}$$

The learning problem is to solve the following optimization problem:

$$\text{minimize}_{\boldsymbol{w}} \quad \sum_{i=1}^{C} p(y=i) \mathbb{E}_{\boldsymbol{x}|y=i}[\log f_i(\boldsymbol{x}, \boldsymbol{w})]. \tag{1}$$

In federated learning, suppose there are N client devices in total. The kth device has $m^{(k)}$ data samples following the data distribution $p^{(k)}$, which is a joint distribution of the samples $\{\boldsymbol{x}, y\}$ on this device. In each round t, K devices are selected (at random in the original FL), each of which downloads the current global model weights $\boldsymbol{w}_{t-1}$ from the server and performs the following stochastic gradient descent (SGD) training locally:

$$\begin{aligned}\boldsymbol{w}_t^{(k)} &= \boldsymbol{w}_{t-1} - \eta \nabla \ell(\boldsymbol{w}_{t-1}) \\ &= \boldsymbol{w}_{t-1} - \eta \sum_{i=1}^{C} p^{(k)}(y=i) \nabla_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{x}|y=i}[\log f_i(\boldsymbol{x}, \boldsymbol{w}_{t-1})],\end{aligned} \tag{2}$$

where η is the learning rate. Note that in such notation with expectations, $\boldsymbol{w}_t^{(k)}$ can be a result one or multiple epochs of local SGD training [1].

Once $\boldsymbol{w}_t^{(k)}$ is obtained, each participating device k reports a model weight difference $\Delta_t^{(k)}$ to the FL server [1], which is defined as

$$\Delta_t^{(k)} := \boldsymbol{w}_t^{(k)} - \boldsymbol{w}_{t-1}^{(k)}.$$

Devices can also upload their local models $\boldsymbol{w}_t^{(k)}$ directly, but transferring $\Delta_t^{(k)}$ is more amenable to compression and communication reduction [4]. After the FL server has collected updates from all K participating devices in round t, it performs the *federated averaging* (FEDAVG) algorithm to update the global model:

$$\Delta_t = \textstyle\sum_{k=1}^{K} m^{(k)} \Delta_t^{(k)} / \sum_{k=1}^{K} m^{(k)},$$

$$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} + \Delta_t.$$

B. The Challenges of Non-IID Data Distribution

FEDAVG has been shown to work well approximating the model trained on centrally collected data, given that the data and label distributions on different devices are IID [1]. However, in reality, data owned by each device are typically non-IID, *i.e.*, $p^{(k)}$ are different on different devices, due to different

[1] Our implementation is available as an open-source GitHub repository, located at `https://github.com/iqua/flsim`.

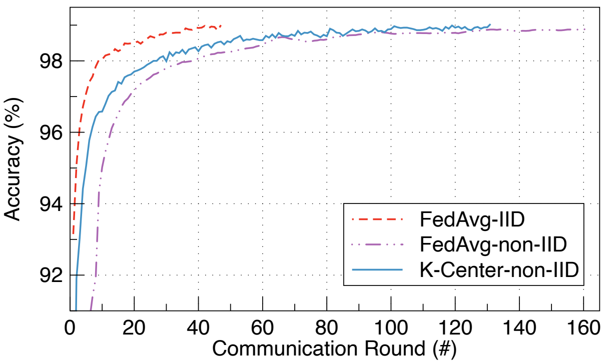


Fig. 1: Training a CNN model on non-IID MNIST data.

user preferences and usage patterns. When data distributions are non-IID, FedAvg is unstable and may even diverge [8].

This is due to the inconsistency between the locally performed SGD algorithm, which aims to minimize the loss value on $m^{(k)}$ local samples on each device and the global objective of minimizing the overall loss on all $\sum_{k=1}^{K} m^{(k)}$ data samples. As we keep fitting models on different devices to heterogeneous local data, the divergence among the weights $\boldsymbol{w}^{(k)}$ of these local models will be accumulated and eventually degrades the performance of learning [8, 10], leading to more communication rounds before training converges. Reducing the number of communication rounds in federated learning is crucially essential for mobile devices, which have a limited computation capacity and communication bandwidth.

We use an experiment to demonstrate that non-IID data may slow down the convergence of federated learning if devices are randomly selected in each round for model aggregation. Instead of random selection, we show that selecting devices *with a clustering algorithm* can help to even out data distribution and speed up convergence.

In our experiment, we train a two-layer CNN model with PyTorch on the MNIST dataset (containing 60,000 samples) until the model achieves a 99% test accuracy. We consider 100 devices in total, each running on a different thread, and we train the CNN model under the IID and non-IID settings, respectively. For the IID setting, the 60,000 samples are randomly distributed among 100 devices, such that each device owns 600 samples. For the non-IID setting, each device still owns 600 samples, yet 80% of which come from a dominant class and the remaining 20% belong to other classes. For example, a “0”-dominated device has 480 data samples with the label “0”, while the remaining 120 data samples have labels evenly distributed among “1” to “9”. In each round, ten devices are selected to participate in federated learning. Note that FL selects only a subset of all devices at each round to balance computation complexity and convergence rate, and to avoid excessively long waiting times for model aggregation [1, 4], which is also discussed in Sec. IV-D.

FedAvg randomly selects ten devices to participate in each round of training. As shown in Fig. 1, FedAvg takes 47 rounds to achieve the target accuracy in the IID setting[2]. However, FedAvg takes 163 rounds to achieve the 99% target accuracy on non-IID data, which more than triples the number of communication rounds.

To reduce the required communication rounds under the non-IID setting, we then tuned the device selection strategy. While in FL, it is impossible to peek into the data distribution of each device, we observe that there is an implicit connection between the data distribution on a device and the model weights trained on that device. Therefore, we may profile each device by analyzing the local model weights after the first round.

We provide some rationale for this observation. According to (2), given the initial global model weights $\boldsymbol{w}_{init}$, the local weights on device k after one epoch of SGD can be represented by

$$\boldsymbol{w}_1^{(k)} = \boldsymbol{w}_{init} - \eta \sum_{i=1}^{C} p^{(k)}(y=i)\nabla_{\boldsymbol{w}}\mathbb{E}_{\boldsymbol{x}|y=i}[\log f_i(\boldsymbol{x}, \boldsymbol{w}_{init})].$$

The weight divergence between device k and device k' can be measured as $\| \boldsymbol{w}_1^{(k')} - \boldsymbol{w}_1^{(k)} \|$, where $k, k' \in [K]$. Then, by using a similar bounding technique adopted in [8], we can derive the following bound for weight divergence in the first round:

$$\| \boldsymbol{w}_1^{(k')} - \boldsymbol{w}_1^{(k)} \| \leqslant \eta g_{max}(\boldsymbol{w}_{init}) \sum_{i=1}^{C} \| p^{(k')}(y=i) - p^{(k)}(y=i) \|,$$

where $g_{max}(\boldsymbol{w}) := \max_{i=1}^{C} \| \nabla_{\boldsymbol{w}}\mathbb{E}_{\boldsymbol{x}|y=i}[\log f_i(\boldsymbol{x}, \boldsymbol{w})] \|$. This implies that even if local models are trained with the same initial weights $\boldsymbol{w}_{init}$, there is an implicit connection between the discrepancy of data distributions on device k and k', which is the term $\sum_{i=1}^{C} \| p^{(k')}(y=i) - p^{(k)}(y=i) \|$, and their weight divergence.

Based on the sights above, we first let each of the 100 devices download the same initial global weights (randomly generated) and perform one epoch of SGD based on local data to get $\boldsymbol{w}_1^{(k)}$, $k = 1, \ldots, 100$. We then apply the K-Center clustering algorithm [11] onto $\{\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_1^{(100)}\}$ to cluster the 100 devices into 10 groups. In each round, we randomly select one device from each group to participate in FL training, which still results in 10 simultaneously participating devices per round. As shown in Fig. 1, the K-Center algorithm takes 131 rounds to reach the target accuracy under the non-IID setting, which is 32 rounds less than the vanilla FedAvg algorithm. This example implies that it is possible to improve the performance of federated learning by carefully selecting the set of participating devices in each round, especially under the non-IID data setting.

C. Deep Reinforcement Learning (DRL)

Reinforcement Learning (RL) is the learning process of an *agent* that acts in corresponding to the *environment* to

[2] The same setting costs FedAvg 50 rounds on TensorFlow in [1].

maximize its *rewards*. The recent success of RL [12, 13] comes from the capability that RL agents learn from the interaction with a dynamic environment. Specifically, at each time step t, the RL agent observes states $\boldsymbol{s}_t$ and performs an action a_t. The state $\boldsymbol{s}_t$ of the environment then transits to $\boldsymbol{s}_{t+1}$, and the agent will receive reward r_t. The state transitions and rewards follow a Markov Decision Process (MDP) [14], which is a discrete time stochastic control process, denoted by a sequence $\boldsymbol{s}_1$, a_1, r_1, $\boldsymbol{s}_2$, $\ldots$, r_{t-1}, $\boldsymbol{s}_t$, a_t, $\ldots$. The objective of RL is to maximize the expectation of the cumulative discounted return $R = \sum_{t=1}^{T} \gamma^{t-1} r_t$, where $\gamma \in (0, 1]$ is a factor discounting future rewards.

The RL agent seeks to learn a cheat sheet that points out the actions leading to the maximum cumulative return under a particular state. Value-based RL algorithms formally uses an action-value function $Q(\boldsymbol{s}_t, a)$ to estimate an expected return starting from state $\boldsymbol{s}_t$:

$$\begin{aligned} Q_\pi(\boldsymbol{s}_t, a) &:= \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k-1} | \boldsymbol{s}_t, a] \\ &= \mathbb{E}_{\boldsymbol{s}_{t+1}, a}[r_t + \gamma Q_\pi(\boldsymbol{s}_{t+1}, a) | \boldsymbol{s}_t, a_t], \end{aligned}$$

where π is the policy mapping from states to probabilities of selecting possible actions. The optimal action-value function $Q^*(\boldsymbol{s}_t, a)$ is the cheat sheet sought by the RL agent, which is defined as the maximum expectation of the cumulative discounted return starting from $\boldsymbol{s}_t$:

$$Q^*(\boldsymbol{s}_t, a) := \mathbb{E}_{\boldsymbol{s}_{t+1}}[r_t + \gamma \max_a Q^*(\boldsymbol{s}_{t+1}, a) | \boldsymbol{s}_t, a]. \quad (3)$$

Then, we could apply function approximation techniques to learn a parameterized value function $Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t)$ approximating the optimal value function $Q^*(\boldsymbol{s}_t, a)$. The one-step lookahead $r_t + \gamma \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{\theta}_t)$ becomes the target that $Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t)$ learns to be. Typically, a deep neural network (DNN) is used to represent the function approximator [15]. The RL learning problem becomes minimizing the MSE loss between the target and the approximator, which is defined as:

$$\ell_t(\boldsymbol{\theta}_t) := (r_t + \gamma \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{\theta}_t) - Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t))^2 \quad (4)$$

III. DRL for Client Selection

We formulate device selection for federated learning as a deep reinforcement learning (DRL) problem. Based on the formulation, we present a federated learning workflow driven by DRL in Sec. III-B. We then introduce the challenges of dealing with high-dimensional models in federated learning. Finally, we describe training the DRL agent with the double Deep Q-learning Network (DDQN) [16].

The per-round FL process can be modeled as a Markov Decision Process (MDP), with state $\boldsymbol{s}$ represented by the global model weights and the model weights of each client device in each round. Given a state, the DRL agent takes an action a to select a subset of devices that perform local training and update the global model. Then a reward signal r is observed, which is a function of the test accuracy achieved by the global model so far on a held-out validation set. The objective is to train the DRL agent to converge to the target accuracy for federated learning as quickly as possible.

In the proposed framework, the agent does not need to collect or check any data samples from mobile devices—only model weights are communicated—thus preserving the same level of privacy as the original FL does. It solely relies on model weight information to determine which device may improve the global model the most, since there is an implicit connection between the data distribution on a device and its local model weights obtained by performing SGD on those data. In fact, in Sec. III-C we will show that even after dimension reduction, the divergence between local model weights is still obvious and contains information to guide device selection.

A. The Agent based on Deep Q-Network

Suppose there is a federated learning job on N available devices with a target accuracy Ω. In each round, using a Deep Q-Network (DQN) [15], K devices are selected to participate in the training. Considering limited available traces from federated learning jobs, DQN can be more efficiently trained and can reuse data more effectively than policy gradient methods and actor-critic methods.

State: Let the state of round t be represented by a vector $\boldsymbol{s}_t = (\boldsymbol{w}_t, \boldsymbol{w}_t^{(1)}, \ldots, \boldsymbol{w}_t^{(N)})$, where $\boldsymbol{w}_t$ denotes the weights of the global model after round t, and $\boldsymbol{w}_t^{(1)}, \ldots, \boldsymbol{w}_t^{(N)}$ are model weights of the N devices, respectively.

The agent collocates with the FL server and maintains a list of model weights $\{\boldsymbol{w}^{(k)} | k \in [N]\}$; a particular $\boldsymbol{w}^{(k)}$ is updated in round t only if device k is selected for training and the resulting $\Delta_t^{(k)}$ is received by the FL server. Therefore, there is no extra communication overhead introduced for the devices.

The resulting state space can be huge, *e.g.*, a CNN model can contain millions of weights. It is challenging to train a DQN with such a large state space. In practice, we propose to apply an effective and lightweight dimension reduction technique on the state space, *i.e.*, on model weights, which will be described in detail in Sec. III-C.

Action: At the beginning of each round t, the agent needs to decide to select which subset of K devices from the N devices. This selection would have resulted in a large action space of size $\binom{N}{K}$, which complicates RL training. We propose a trick to keep the action space small while still leveraging the intelligent control provided by the DRL agent.

In particular, the agent is trained by selecting only one out of N devices to participate in FL per round based on DQN, while in testing and application the agent will sample a batch of top-K clients to participate in FL. That is, the DQN agent learns an approximator of the optimal action-value function $Q^*(\boldsymbol{s}_t, a)$ through a neural network, which estimates the action that maximizes the expected return starting from $\boldsymbol{s}_t$. The action space is thus reduced to $\{1, 2, \ldots, N\}$, where $a = i$ means that device i is selected to participate in FL.

Once DQN has been trained to approximate $Q^*(\boldsymbol{s}, a)$, during testing, in round t the DQN agent will compute

$\{Q^*(\boldsymbol{s}_t, a) | a \in [N]\}$ for all N actions. Each action-value indicates the maximum expected return that the agent can get by selecting a particular action a at state $\boldsymbol{s}_t$. Then we select the K devices, each corresponding to a different action a, that lead to the top-K values of $Q^*(\boldsymbol{s_t}, a)$.

Reward: We set the reward observed at the end of each round t to be $r_t = \Xi^{(\omega_t - \Omega)} - 1$, $t = 1, \ldots, T$, where ω_t is the testing accuracy achieved by the global model on the held-out validation set after round t, Ω is the target accuracy, and Ξ is a positive constant that ensures that r_t grows exponentially with the testing accuracy ω_t. We have $r_t \in (-1, 0]$, since $0 \leqslant \omega_t \leqslant \Omega \leqslant 1$. The federated learning stops when $\omega_t = \Omega$, at which point r_t reaches its maximum value of 0.

The DQN agent is trained to maximize the expectation of the *cumulative discounted reward* given by

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t = \sum_{t=1}^{T} \gamma^{t-1} (\Xi^{(\omega_t - \Omega)} - 1),$$

where $\gamma \in (0, 1]$ is a factor discounting future rewards.

We now explain the motivations behind the two terms $\Xi^{(\omega_t - \Omega)}$ and -1 in r_t. The first term, $\Xi^{(\omega_t - \Omega)}$, incentivizes the agent to select devices that achieve a higher test accuracy ω_t. Ξ controls how fast reward r_t grows with ω_t. In general, as ML training proceeds, the model accuracy will increase at a slower pace, which means $|\omega_t - \omega_{t-1}|$ decreases as round t increases. Therefore, we use an exponential term to amplify the marginal accuracy increase as FL progresses into later stages. Ξ is set to 64 in our experiments.

The second term, -1, encourages the agent to complete training in fewer rounds, because the more rounds it takes, the less cumulative reward the agent will receive.

B. *Workflow*

Fig. 2 shows how our system FAVOR performs federated learning with a DRL agent selecting devices in each round, following the steps below:

Step 1: All N eligible devices check in with the FL server.

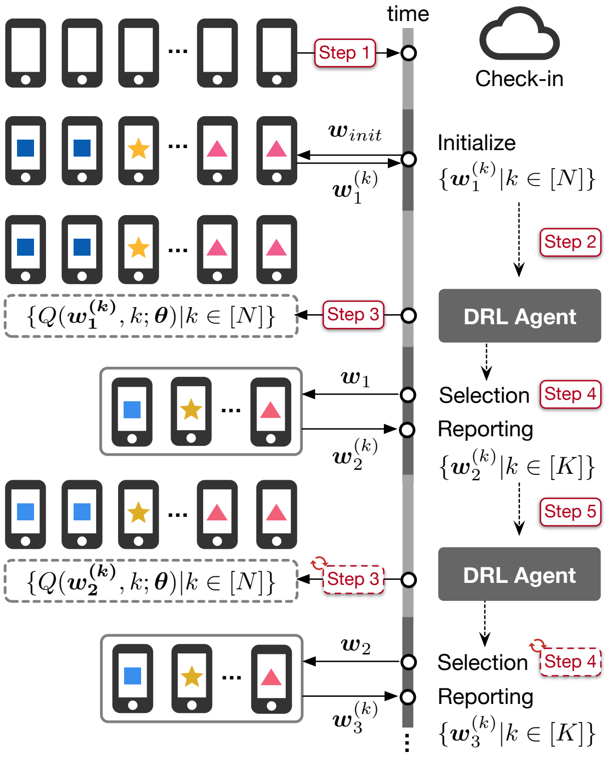
Step 2: Each device downloads the initial random model weights $\boldsymbol{w}_{init}$ from the server, performs local SGD training for one epoch, and returns the resulting model weights $\{\boldsymbol{w}_1^{(k)}, k \in [N]\}$ to the FL server.

Step 3: In round t, where $t = 1, 2, \ldots$, upon receiving the uploaded local weights, the corresponding copies of local model weights stored on the server are updated. The DQN agent computes $Q(\boldsymbol{s}_t, a; \boldsymbol{\theta})$ for all devices $a = 1, \ldots, N$.

Step 4: The DQN agent selects K devices corresponding to the top-K values of $Q(\boldsymbol{s}_t, a; \boldsymbol{\theta})$, $a = 1, \ldots, N$. The selected K devices download the latest global model weights $\boldsymbol{w}_t$ and perform one epoch of SGD locally to obtain $\{\boldsymbol{w}_{t+1}^{(k)} | k \in [K]\}$.

Step 5: $\{\boldsymbol{w}_{t+1}^{(k)} | k \in [K]\}$ are reported (uploaded) to the server to compute $\boldsymbol{w}_{t+1}$ based on FEDAVG. Move into round $t + 1$ and repeat Steps 3-5.

Steps 3-5 will be repeated until completion, *e.g.*, until a target accuracy is reached or after a certain number of rounds.



time
Step 1
Check-in
$\boldsymbol{w}_{init}$
$\boldsymbol{w}_1^{(k)}$
Initialize
$\{\boldsymbol{w}_1^{(k)} | k \in [N]\}$
Step 2
$\{Q(\boldsymbol{w_1^{(k)}}, k; \boldsymbol{\theta}) | k \in [N]\}$
Step 3
DRL Agent
$\boldsymbol{w}_1$
Selection
Step 4
Reporting
$\boldsymbol{w}_2^{(k)}$
$\{\boldsymbol{w}_2^{(k)} | k \in [K]\}$
Step 5
$\{Q(\boldsymbol{w_2^{(k)}}, k; \boldsymbol{\theta}) | k \in [N]\}$
Step 3
DRL Agent
$\boldsymbol{w}_2$
Selection
Step 4
Reporting
$\boldsymbol{w}_3^{(k)}$
$\{\boldsymbol{w}_3^{(k)} | k \in [K]\}$

Fig. 2: The federated learning workflow with FAVOR.

Note that the above workflow can be easily tweaked to let each selected client upload the model difference Δ (as compared to the last version of the local model) instead of the new local model itself, without changing the underlying logic. The FL server can always use the Δ to reconstruct a copy of the corresponding local model stored on it.

When a device k has new training data, it will request the global model weights $\boldsymbol{w}_t$ and perform local SGD training for one epoch as Step 2, which generates new local model weights $\boldsymbol{w}_t^{(k)}$. Then the device pushes $\boldsymbol{w}_t^{(k)}$ to the FL server, and the FL server updates the state $\boldsymbol{s}_t$ to make the DRL agent aware of the device with updated data. It should be noted that data on each device remain unchanged during the DRL training to ensure that the training follows the Markov Decision Process.

The overhead introduced is minimum since no extra computation/communication overhead is added to devices. The only overhead is that now the FL server needs to store a copy of the latest local model weights from each device in order to form the state $\boldsymbol{s}_t$. However, the FL server is deployed in cloud, where provisions sufficient on-demand computational and storage capacity .

C. *Dimension Reduction*

One issue of the proposed DQN agent is that it uses the weights of the global model and all local models as the state, which leads to a large state space. Many deep neural networks (*e.g.*, CNN) have millions of weights, making it challenging to train the DQN agent with such a high dimensional state space. To reduce the dimension of the state space, we propose

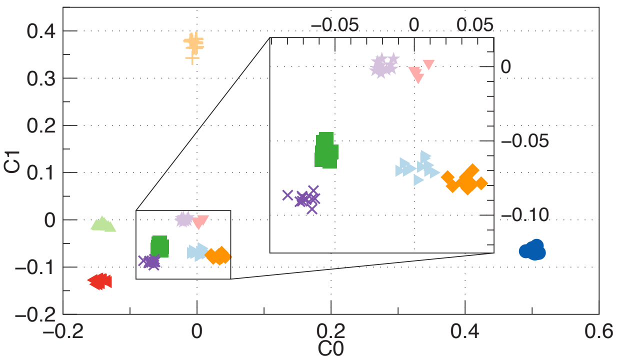


Fig. 3: PCA of CNN weights on the MNIST dataset.

to apply principle component analysis (PCA) to model weights and use the compressed model weights to represent states instead.

Specifically, we compute the PCA loading vectors of different principal components only based on local model weights for $t = 1$, *i.e.*, $\{\boldsymbol{w}_1^{(k)}, k \in [N]\}$, obtained in Step 2. In the subsequent rounds $t = 2, 3, \ldots$, such loading vectors are reused to obtain first several principal components of $\{\boldsymbol{w}_t^{(k)}, k \in [N]\}$, through linear transformation, without fitting the PCA model again. Therefore, the overhead to compress model weights in subsequent rounds is negligibly small.

We demonstrate the effectiveness of using PCA-compressed model weights to differentiate between data distributions through a simple experiment. Consider a two-layer CNN model (with 431,080 model weights in total) to be trained with federated learning on 100 devices, running PyTorch, on the MNIST dataset. Data samples are distributed in the same way as the experiment in Sec. II: each client has 80% of its data samples belonging to a dominant class, while the remaining 20% of its samples have random labels. After five epochs of local SGD training in Round 1, we project the model weight vectors of the 100 devices, $\{\boldsymbol{w}_1^{(k)}, k \in [N]\}$, onto a two-dimensional space of the first and second principal components.

As shown in Fig. 3, different shapes (or colors) indicate local models trained on devices with different dominant labels. For example, all the yellow "+" signs denote the compressed local models in Round 1 from those devices with a dominant label "6". Even reduced from 431,080 dimensions to only two dimensions and even at Round 1, we can observe a clustering effecting of local models according to their dominant labels. Therefore, in the evaluation in Sec. IV, we use PCA-compressed model weights as states to enable efficient training of the DQN agent.

D. Training the Agent with Double DQN

We propose to use the double deep Q-learning network (DDQN) to learn the function $Q^*(\boldsymbol{s}_t, a)$. Q-learning provides a value estimation for each potential action a at state $\boldsymbol{s}_t$, based on which devices are selected. However, the original Q-learning algorithms can be unstable since they indirectly

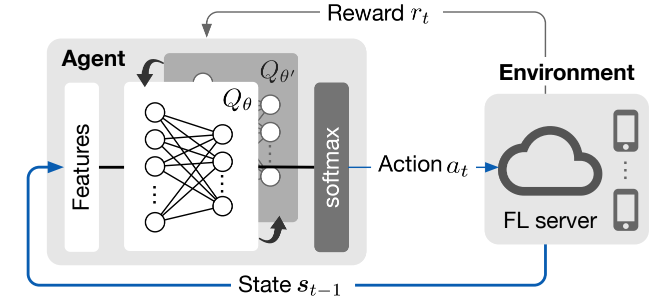


Fig. 4: The DDQN Agent interacting with the FL server.

optimize the agent performance by learning an approximator $Q(\boldsymbol{s}, a; \boldsymbol{\theta}_t)$ to the optimal action-value function $Q^*(\boldsymbol{s}, a)$. DDQN adds another value function $Q(\boldsymbol{s}, a; \boldsymbol{\theta'}_t)$ to stabilize the action-value function estimation. The idea behind DDQN is that the network is frozen every M updates. DDQN adds stability to the action-value evaluation, which is less prone to "jittering."

To train the DRL agent, the FL server first performs random device selection to initialize the states. As shown in Fig. 4, the states are fed into the one of the double DQNs $Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t)$. The DQN generates an action a to select a device for the FL server. After several rounds of FL training, the DRL agent has sampled a few action-state pairs, with which the agent learns to solve the (4) as:

$$\ell_t(\boldsymbol{\theta}_t) = (Y_t^{DoubleQ} - Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t))^2,$$

where $Y_t^{DoubleQ}$ is the target at round t defined as

$$\begin{aligned} Y_t^{DoubleQ} &:= r_t + \gamma \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{\theta}_t) &(5)\\ &= r_t + \gamma Q(\boldsymbol{s}_t, \operatorname{argmax}_a Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t); \boldsymbol{\theta'}_t). &(6)\end{aligned}$$

(6) uses two acton-value functions to update $Y_t^{DoubleQ}$, in which $\boldsymbol{\theta}_t$ is the online parameters updated per time step, and the $\boldsymbol{\theta'}_t$ is the frozen parameters to add stability to action-value estimation. The action-value function $Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{\theta}_t)$ is updated to minimize $\ell_t(\boldsymbol{\theta}_t)$ by gradient descent, *i.e.*,

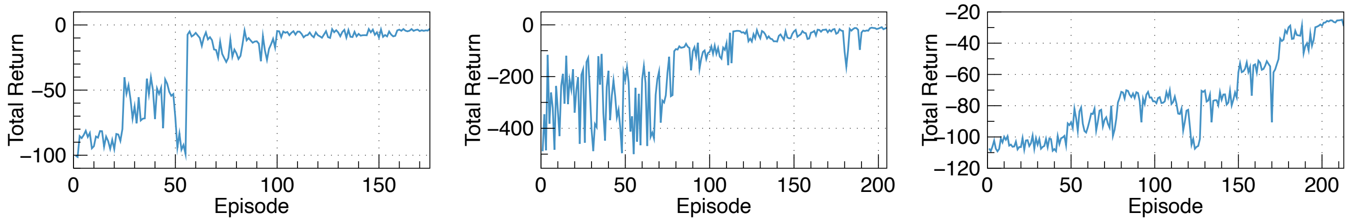
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{DoubleQ} - Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}_t),$$

where α is a scalar step size.

IV. Evaluation

We have implemented Favor with PyTorch in around 2000 lines of code, which we have released as an open-source project. With the Python `threading` library, Favor can simulate a large number of devices with lightweight threads, each running real-world PyTorch models.

We evaluated Favor by training popular CNN models on three benchmark datasets: MNIST, FashionMNIST, and CIFAR-10, with FedAvg and K-Center as the groups of comparison. We evaluated the accuracy of the trained models using the testing set from each dataset. Our experimental results show that Favor can reduce the communication rounds by up to 49% on the MNIST, up to 23% on FashionMNIST,



(a) DRL training on the MNIST.

(b) DRL training on FashionMNIST.

(c) DRL training on CIFAR-10.

Fig. 5: Training the DRL agent.

and up to 42% on CIFAR-10, compared to the FEDAVG algorithm. We briefly describe our methodology and settings as follows.

Datasets and models: We explored different combinations of hyper-parameters for the CNN models on different datasets and chose the hyper-parameters leading to the best performance of FEDAVG.

- **MNIST**. We train a CNN model with two 5×5 convolution layers. The first layer has 20 output channels and the second has 50, with each layer followed by 2×2 max pooling. On each device, the batch size is ten and the epoch number is five.
- **FashionMNIST**. We train a CNN model with two 5×5 convolution layers. The first layer has 16 output channels and the second has 32, with each layer followed by 2×2 max pooling. On each device, the batch size is 100 and the epoch number is five.
- **CIFAR-10**. We train a CNN model with two 5×5 convolution layers. The first layer has six output channels and the second having 16, with each layer followed by 2×2 max pooling. On each device, the batch size is 50 and the epoch number is five.

Performance metrics: In federated learning, due to the limited computation capacity and network bandwidth of mobile devices, reducing the number of communication rounds is crucially important. Thus, we use the number of communication rounds as the performance metric of FAVOR.

A. Training the DRL agent

We train the DRL agent on different datasets with 100 available devices. The DDQN model in the DRL agent consists of two two-layer MLP networks, with 512 hidden states. The input size is 10,100, where we have 101 model weights (*i.e.*, the global weights $\boldsymbol{w}$ and the local weights $\{\boldsymbol{w}^{(k)}|k \in [100]\}$ from 100 devices) reduced to 100 dimensions. The output size of the second layer is 100. Each output passing through a softmax layer becomes the probability of selecting the particular device. We train the DRL agent on an AWS EC2 instance `p2.2xlarge` with a K80 GPU. The DDQN model is lightweight, and each training iteration takes seconds on GPU. Reducing model weights from 431,080 to 100 dimensions takes minutes by `sklearn.decomposition.PCA`.

Fig. 5 plots the training progress of the DRL agent on three FL tasks. An episode starts at the initialization of a federated learning job and ends when the job converges to the target accuracy Ω. The total return is the cumulative discounted reward R obtained in one episode. The target accuracy Ω is set to 99% for training on the MNIST, 85% for training on FashionMNIST, and 55% for training on CIFAR-10.

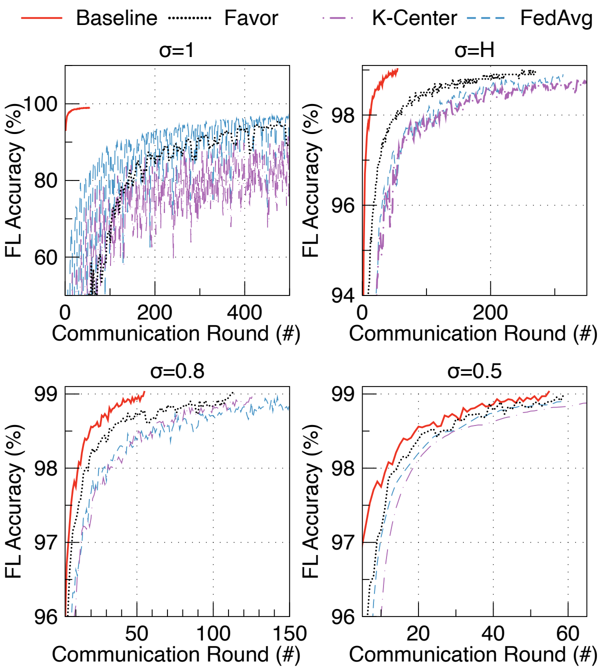


Fig. 6: Accuracy *v.s.* communication rounds on different levels of non-IID MNIST data.

B. Different Levels of Non-IID Data

We investigate different levels of non-IID data to testify the efficiency of FAVOR. We include the performance of FEDAVG and K-Center as a comparison. At each round, the number of selected devices K is set to 10, as in [1].

We use σ to denote the four different levels of non-IID data: $\sigma = 1.0$ indicating that data on each device only belong to one label, $\sigma = 0.8$ indicating that 80% of the data belong to one label and the remaining 20% data belong to other labels, $\sigma = 0.5$ indicating that 50% of the data belong to one label and the remaining 50% data belong to other labels, $\sigma = 0$ indicating the data one each device is IID and $\sigma = H$ indicating that data on each device evenly belong to two labels. We plot the test-set accuracy *v.s.* the communication rounds of federated learning on the three benchmarks in Fig. 6, Fig. 7, and Fig. 8.

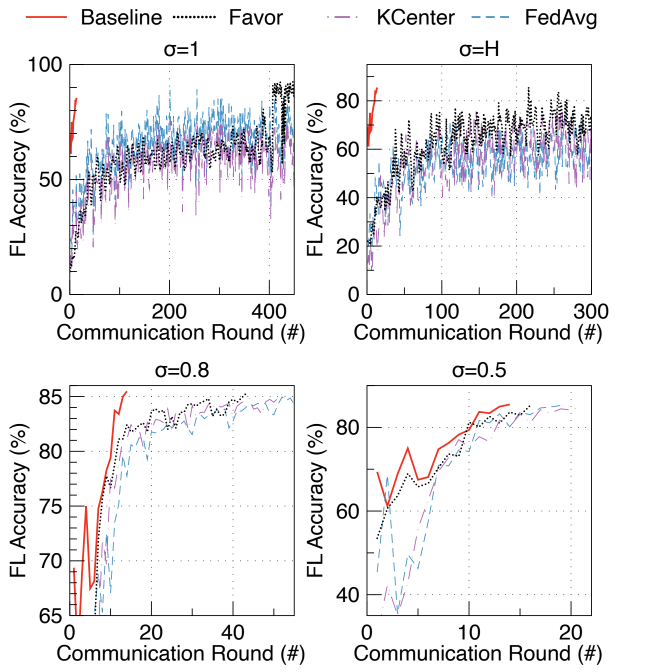


Fig. 7: Accuracy *v.s.* communication rounds on different levels of non-IID FashionMNIST data.

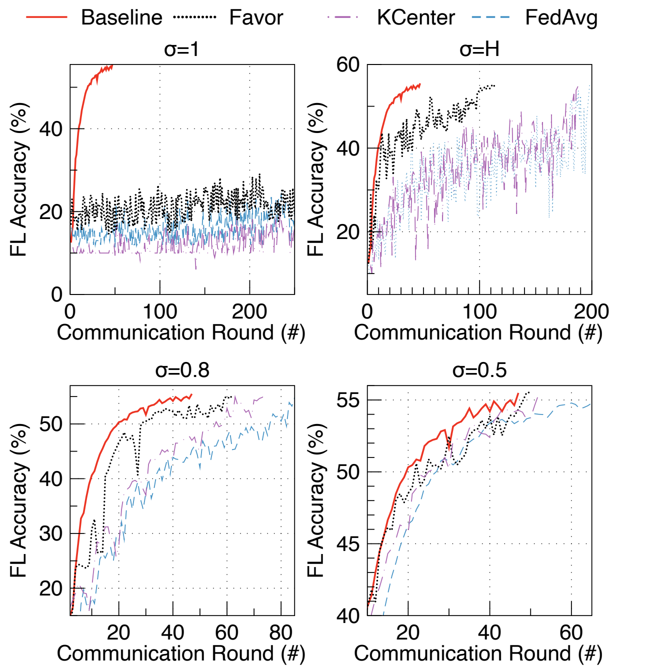


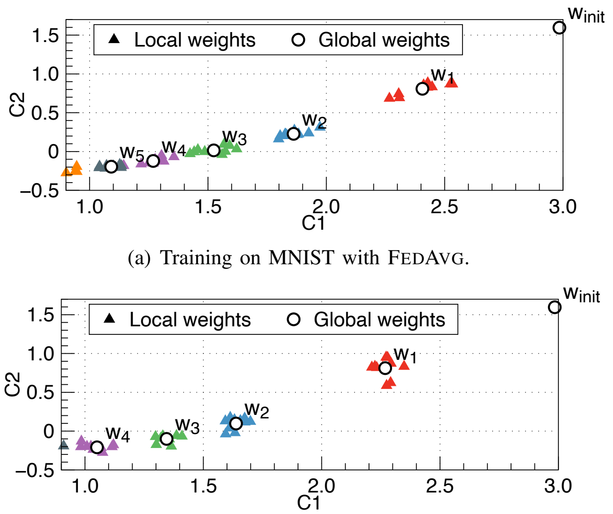
Fig. 8: Accuracy *v.s.* communication rounds on different levels of non-IID CIFAR-10 data.

Each entry in Table I shows the number of communication rounds necessary to achieve a test-set accuracy of 99% for the CNN on MNIST, 85% for FashionMNIST, and 55% for CIFAR-10. It should be noted that the italic numbers indicate that the model converges to a lower accuracy than the test-set accuracy. The model on MNIST with data distribution of $\sigma = 1.0$ converges to a test-set accuracy of 96%. The models trained on FashionMNIST with data distribution of $\sigma = 1.0$ and $\sigma = H$ both converge to a test-set accuracy of 80%. The model trained on CIFAR-10 with data distribution of $\sigma = 1.0$ converges to a test-set accuracy of 50%.

Our experimental results show there is no guarantee that K-Center can always outperform FEDAVG. Devices selected from the K-Center clusters may introduce more bias to federated learning than devices selected randomly by FEDAVG. However, FAVOR has reduced the number of communication rounds by up to 49% on the MNIST, up to 23% on FashionMNIST, and up to 42% on CIFAR-10, compared to the FEDAVG algorithm.

	σ	MNIST	FashionMNIST	CIFAR-10
FEDAVG	0 (IID)	55	14	47
FEDAVG	1.0	*1517*	*1811*	*1714*
K-Center	1.0	*1684*	*2132*	*1871*
FAVOR	1.0	*1232*	*1497*	*1383*
FEDAVG	H	313	*1340*	198
K-Center	H	421	*1593*	188
FAVOR	H	272	*1134*	114
FEDAVG	0.8	221	52	87
K-Center	0.8	126	62	74
FAVOR	0.8	113	43	61
FEDAVG	0.5	59	19	67
K-Center	0.5	67	21	52
FAVOR	0.5	59	16	50

TABLE I: The number of communication rounds to reach a target accuracy for FAVOR *v.s.* FEDAVG and K-Center.



(a) Training on MNIST with FEDAVG.

(b) Training on MNIST with FAVOR.

Fig. 9: PCA on model weights of FL training with MNIST. $w_1, w_2 \ldots, w_5$ are the global model weights at Round 1-5.

C. Device Selection and Weight Updates

We save the global model weights and local model weights per round when we train the two-layer CNN on MNIST with $\sigma = 0.8$. The saved model weights are reduced to two-

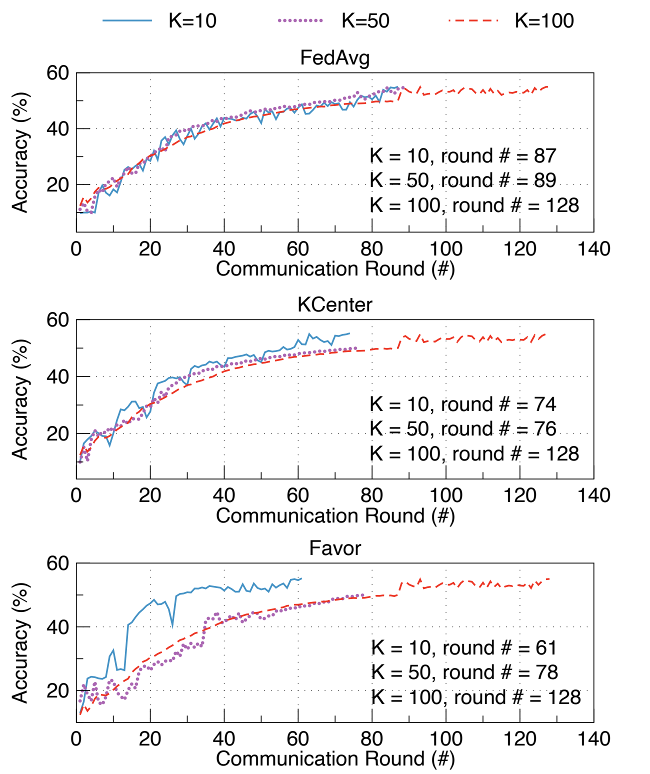


Fig. 10: FL training on CIFAR-10 with different levels of parallelism.

dimension vectors by PCA and plotted in Fig. 9. By examining the model weights trained with FEDAVG and FAVOR, respectively, Fig. 9 shows that FAVOR updates the global model with a larger weight update Δ than FEDAVG in early rounds, which leads to a faster convergence speed.

D. Increasing Parallelism

We also studied the performance of FAVOR on different numbers of selected devices. The CIFAR-10 dataset is distributed to 100 devices with the non-IID level at $\sigma = 0.8$. We apply FAVOR, FEDAVG, and K-Center to train the same CNN on CIFAR-10 separately. The number of selected devices K is set to 10, 50, and 100 to study the performance of federated learning with different parallelism. Fig. 10 shows that increasing the parallelism does not reduce the number of communication rounds and even increases the number of communication rounds.

V. Related Work

Federated learning allows machine learning models to be trained on mobile devices in a distributed fashion without violating user privacy. Existing studies on federated learning have mostly focused on improving its efficiency. We classify the existing literature into the following two categories:

Communication efficiency. Mobile devices typically have unstable and expensive connections, and existing works have attempted to improve the communication efficiency of federated learning. Konečnỳ *et al.* [6, 7] proposed structured and sketched updates to decrease the completion time of each communication round. Bonawitz *et al.* [3] developed a secure aggregation protocol that enables a server to perform the computation of high-dimensional data from the devices, which is widely deployed in production environments [4, 17]. McMahan *et al.* [1] presented the Federated Averaging (FEDAVG) algorithm that allows devices to perform local training of multiple epochs, which further reduces the number of communication rounds by averaging model weights from the client devices. Nishio *et al.* [18] proposed a resource-aware selection algorithm that maximizes the number of participating devices in each round. Sattler *et al.* [9] proposed a compression framework, Sparse Ternary Compression (STC), that reduces the communication costs and remains robust to non-IID data. FAVOR applies DRL to select the best subset of participating devices to minimize the number of communication rounds, which is orthogonal to these studies on communication efficiency.

Sample efficiency. Unlike centralized machine learning, federated learning performs training on non-IID data on devices. Zhao *et al.* [8] presented a mathematical demonstration to show that non-IID data reduces the accuracy of federated learning by a substantial margin, and proposed to push a small set of uniform distributed data to participating devices. Downloading extra data further increases communication cost and computation workload for the devices. Mehryar *et al.* [10] proposed an agnostic federated learning framework for fairness to avoid biases due to non-IID data from the devices. In contrast, FAVOR is the first work that counterbalances the bias from different non-IID data by dynamically constructing the subset of participating devices with DRL techniques.

VI. Concluding Remarks

In this paper, we presented our design and implementation of FAVOR, an experience-driven federated learning framework that performs active device selection to minimize the communication rounds of FL training. We argue that non-IID data exacerbates the divergence of model weights on participating devices, and increases the number of communication rounds of federated learning by a substantial margin. With both mathematical demonstrations and empirical studies, we found the implicit connection between model weights and the distribution of data that the model is trained on. We proposed to actively select a specific subset of devices to participate in training at each round, in order to counterbalance the bias introduced by non-IID data on each device and to speedup FL training by minimizing the number of communication rounds. In particular, we designed a DRL-based agent that applies the DDQN algorithm to select the best subset of devices to achieve our objectives. We have implemented an open-source prototype of FAVOR with PyTorch in more than 2K lines of code and evaluated it with a variety of ML models. An extensive comparison with FL training jobs by FEDAVG has shown that FL training with FAVOR has reduced the number of communication rounds by up to 49% on the MNIST dataset, up to 23% on FashionMNIST, and up to 42% on CIFAR-10.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. the Artificial Intelligence and Statistics Conference (AISTATS)*, 2017.

[2] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated Multi-Task Learning," in *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical Secure Aggregation for Privacy-Preserving Machine Learning," in *Proc. the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba *et al.*, "Towards Federated Learning at Scale: System Design," in *Proc. the Conference on Systems and Machine Learning (SysML)*, 2019.

[5] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," in *Proc. the International Conference on Learning Representations (ICLR)*, 2018.

[6] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," in *Proc. the NIPS Workshop on Optimization for Machine Learning (OPT)*, 2015.

[7] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," in *Proc. the NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[8] 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.

[9] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning from Non-IID Data," *arXiv preprint arXiv:1903.02891*, 2019.

[10] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic Federated Learning," in *International Conference on Machine Learning (ICML)*, 2019.

[11] O. Sener and S. Savarese, "Active Learning for Convolutional Neural Networks: A Core-Set Approach," in *Proc. the International Conference on Learning Representations (ICLR)*, 2018.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proc. the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016.

[13] A. Mirhoseini, H. Pham, Q. Le, M. Norouzi, S. Bengio, B. Steiner, Y. Zhou, N. Kumar, R. Larsen, and J. Dean, "Device Placement Optimization with Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2017.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[15] M. Volodymyr, K. Kavukcuoglu, D. Silver, A. Graves, and I. Antonoglou, "Playing Atari with Deep Reinforcement Learning," in *Proc. NIPS Deep Learning Workshop*, 2013.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proc. the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[17] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied Federated Learning: Improving Google Keyboard Query Suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[18] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *Proc. the IEEE International Conference on Communications (ICC)*, 2019.