Reproduction and Evaluation for Optimal Device Selection in Federated Learning using Deep Q-learning

Tian Liu
UIN:525004380
Texas A&M University
TX, USA

Cheng Niu
UIN:532008693
Texas A&M University
TX, USA

Yuting Cai
UIN:232003637
Texas A&M University
TX, USA

Abstract

In this final course project for CSCE 689 Deep Reinforcement Learning, we tried to reproduce the work done by Wang et al [1], which applied deep reinforcement learning to optimize the federated learning (FL) on Non-IID Data. Specifically, Wang et al. formulated the device selection problem during FL as a Markov Decision Problem and applied the DQN model to select the participating devices in each communication round to achieve faster convergence and less communication cost. We used the given *flsim* git repository as a starting framework for the Federated Learning system and implemented the Double DQN model based on the code given in the homework. We implemented the PCA for state compression and tested the original reward function as proposed in the paper.

However, the testing result in the MNIST dataset with different device settings (100 clients and 20 clients) shows that our implemented DQN model cannot reproduce the same training performance as shown in the paper. Based on discussions with other researchers online, we found that the reward function as shown in the paper is not correct. We further proposed new reward function and analyzed the reasons why DQN is not appropriate for the device selection problem during FL which has strong dependency between actions. Similar to many other researchers' experiences, we are highly doubtful of the results shown in the paper. And the results of our experiments indicate that DQN did not learn how to select the device.

Our GitHub repository is https://github.com/tian1327/flsim_dqn. Our 5-minute Youtube video presentation is available here https://youtu.be/ZNkAfigHkN0.

Keywords: federated learning, reinforcement learning, deep Q-learning

1 Introduction and Motivation

With recent fast development in machine learning, there have been many deep learning applications in people's life, such as human activity recognition through smart phone or smart watch, next word prediction in keyboard typing, voice recognition. The training of the deep learning models is usually conducted in the cloud with powerful GPUs and requires each end devices to send their local data to the central server. However, sending local data to the cloud has raised significant privacy concern among users.

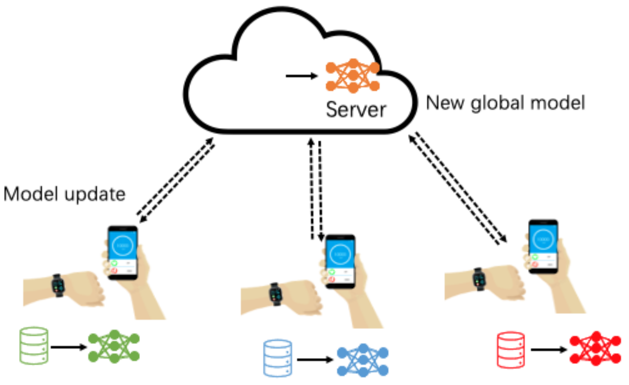


Figure 1. Illustration of Federated Learning System. (By Ouyang et al. [3])

As a solution, Federated learning is a new machine learning paradigm which enables multiple devices collaboratively learn from each other while preserving the privacy of user data at each device. As shown in Figure 1, each end devices will conduct local training based on their local data and then send their local model update instead of local data to the central server. The server then aggregates the model updates it received and send back the new updated model to end devices. The end devices will conduct more local training using the new model received. Each communication between the end devices and server is referred to as a round. The process is repeated until the model converges, thus resulting multiple rounds. In such a manner, the privacy of local data on each end devices can be preserved.

1.1 Device Selection in Federated Learning System

However, in practice when there are millions of end devices which have different network conditions, having all devices to participate in the training will lead to long wait time and thus impractical. A common practice is to randomly sample a certain number of devices in each communication round to participate the training. However, the data at each device is usually not independent and identically distributed (non-IID). Randomly selecting participating devices will introduce unbalanced dataset leading to poor model performance and slow convergence i.e. more communication rounds [2].

1.2 Motivation of using DRL to optimize device selection

To optimize the performance of federated learning on non-IID data, Wang et al. [1] proposed applying deep reinforcement learning to learn to optimally select the participating devices for each training rounds to speed up the convergence. In particular, with an observation that there are an implicit connection between the distribution of training data on local device and the model weights trained based the local data, Wang et al. proposed to adopt reinforcement learning model to learn to select a subset of devices in each communication round to maximize the reward which encourages the increase of validation accuracy and penalizes the use of more communication rounds.

1.3 Our Contributions in this project

1. We successfully implemented the DQN model for device selection based on the paper.
2. We implemented the principle component analysis (PCA) for state space compression based on the local weights of all clients and successfully reproduced the clustered clients based on their local weight updates.
3. We tested our implemented DQN model for various device settings (100 clients select 10 each round, 20 clients select 4 each round) on the MNIST dataset. We analyzed its performance and why the DQN did not work.
4. We proposed new reward function based on our analysis that the original reward function is incorrect. We tested the performance of new reward function in different settings.
5. We successfully reproduced several figures in the paper, such as Figure 3 and Figure 4.
6. We fixed various bugs in the original *flsim* system, such as missing members in class, various typos, etc.
7. We shared our results and findings with other researchers who are also struggled with implementing DQN in FL (https://github.com/iQua/flsim/issues/7).

1.4 Our Reference Code

1. We implemented the DQN server based on the provided *flsim* git repository at https://github.com/iQua/flsim. Notice that this original git repository only provides the simulation framework for the federated learning system. It does not contain the DQN server used in the paper, for some unknown/suspicious reasons.
2. We implemented the DQN training framework based on the DQN homework code given in CSCE689 Deep Reinforcement Learning course.

2 MDP Definition

Here we explain why the device selection in federated learning framework can be modeled as a Markov Decision Process following the work by Wang et al [1].

2.1 Why the device selection problem is sequential?

The device selection problem can be defined as selecting K devices from total N available devices to participate in each round of training for the federated learning job. Figure 2 shows the workflow of using Reinforcement Learning to select k devices for each round of the training.

After all N devices downloading the initial random model weights W_{init} and training to return the result model weights $w_1^{(k)}, k \in [N]$ to the FL server, the FL server generate a $Q(s_t, a; \theta)$ for all devices.

Then the agent selects K devices based on the top-K values from the generation. These devices download the latest global weights w_t and train to obtain result model weights $w_2^{(K)}, k \in [K]$, the FL server generate new $Q(s_{t+1}, a; \theta)$ from report of devices, and then repeat this process sequentially.

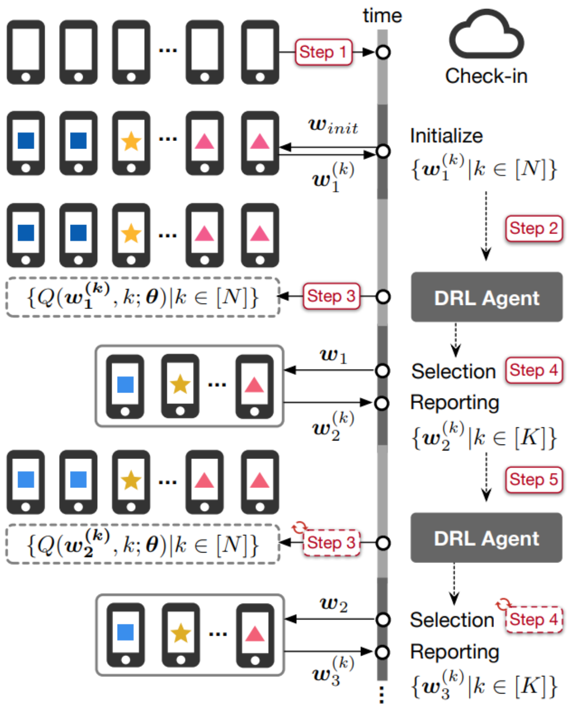


Figure 2. Workflow of device selection problem. (By Wang et al. [1])

2.2 State space

In the device selection problem, the state space S is the global model weights and the model weights of each client device in each round. The vector $S_t = (w_t, w_t^{(1)}, \ldots, w_t^{(N)})$,where w_t denotes the weights of the global model after round t, and $w_t^{(1)}, \ldots, w_t^{(N)}$ denotes the model weights of the N devices respectively. Because the weights have large dimension which makes the state space very large, in the paper, Wang et al. [1] applied principle component analysis (PCA) to reduce the dimensionality down to 100. Thus, the final state space is a vector of 100 dimensions.

2.3 Action space

In the device selection problem, the action is to select subset of K devices from N devices. However, this selection would have resulted in a large action space of size $\binom{K}{N}$, it will complicates the Reinforcement Learning training, so we need to take a trick to reduce the action space size.

We train the agent by select only one out of N devices to participate in FL per round, while in testing and application, the agent will sample a batch of top-K clients to participate in FL. Now the action space is thus reduced to $1, 2, \ldots, N$, which means select device i to participate in FL. The other clients will be selected through the top-K values of $Q^*(s_t, a)$ after our training completion.

2.4 Transition function

In the device selection problem, the transition function is stochastic. For every training round, different K devices are selected to learn the new model weights, and then they conduct new local training with the new model and send updates to the server. Thus the new state is updated again with new local model update received from K devices.

2.5 Reward function

In the device selection problem, we define the reward function at the end of each round t is $r_t = \Xi^{(\omega_t - \Omega)} - 1, t = 1, \ldots, T$ The 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.

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, Ξ is a positive constant that ensures that r_t grows exponentially to amplify the marginal accuracy increase as FL progresses into later stages.

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

Thus, the reward function aims to achieve target accuracy in the least communication rounds.

2.6 RL algorithms to use and baselines

In the device selection problem, we use the double Deep Q-learning Network(DDQN) to learn the function $Q^*(s_t, a)$.

The Q-learning algorithm provides a value estimation for each potential action a at state s_t, based on which devices are selected. Considering limited available traces from federated learning jobs, the DQN can be more efficiently trained and can reuse data more effectively than policy gradient methods and actor-critic methods.

However, the original Q-learning algorithms can be unstable since they indirectly optimize the agent performance by learning an approximator $Q(s, a; \theta_t)$ to the optimal action-value function $Q^*(s, a)$. DDQN adds another value function $Q(s, a; \theta'_t)$ to stabilize the action-value function estimation.

If time permits, we will try to implement other RL algorithms such as policy gradient or actor-critic methods.

2.7 Stretch goal

One of our stretch goals is to try to understand how the given code implements the federated learning system since none of us have experience with FL coding before. Another stretch goal is to implement DQN on top of the FL system and try to reproduce the results reported in the paper.

3 Related Works

In this section, we introduced previous work on device selection problems in Federated Learning framework and different reinforcement learning models. And we discuss how our work of using reinforcement learning differs from other works.

3.1 Device Selection in Federated Learning System

Because of the large amounts of participation devices in Federated Learning system, the limited network connectivity and bandwidth, as well as the long training time on the server, in practice it is infeasible to have all devices participate in every round of training is Federated Learning Systems. To address the device selection problem, McMahan et al.[4]first proposed FEDAVG algorithm which selects a random fraction of total devices to participate in each round of training. Specifically, at each round, server will randomly select k devices and send the global model weights to each client to conduct local training based on their local data. Then these k device will send their local weight updates back to the server. And the server will take the average of these weight updates. The process repeats until the model converges. However, the randomly selected local datasets may not have the same distribution as the true data distribution. And each local model could be significantly different from each other, leading to slow convergence and model accuracy reduction[2].

To address the non-IID problem, Ouyang et al.[3] proposed the ClusterFL, which groups the devices into clusters and conducts the average within each cluster. The key observation of the ClusterFL is that there are intrinsic similarities between different devices due to the subject's biological features, physical environments, and sensor biases. To be specific, the server will cluster the devices based on the KL divergence of the local model updates. And only devices within the same cluster are averaging their model weights. The paper also proposed to drop nodes that have less correlation to other nodes in the same cluster and to drop straggler nodes of each cluster to reduce the communication cost. Results show that ClusterFL achieved much faster convergence and reduced communication rounds significantly.

Nishio et al.[5] proposed FedCS to address the inefficient training among devices with heterogeneous resources. Essentially, FedCS selects the devices based on their resource conditions to maximize the number of participating devices in each round of training. The selection of devices is based on the estimated time for each device to finish its distribution, schedule update, and upload process based on their computation resources and network bandwidth.

Cho et al.[6] proposed FLAME, a user-centered FL training by leveraging the time alignment across the multiple devices owned by the same user. FLAME also conducts accuracy and efficiency-aware device selection which achieves higher accuracy, greater energy efficiency, and faster convergence. Cho et al.[7] also proposed *POWER-OF-CHOICE*, which adopts higher sampling probability for clients with higher loss in each round to achieve faster convergence.

3.2 Reinforcement Learning Models

3.2.1 Model Based On Deep Q-Network. Q-learning is a kind of model-free reinforcement learning, which also be cast as a kind of asynchronous dynamic programming. [8]With Q-learning, agents have the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions.

Deep Q-learning is an extension of the classical Q-learning algorithm that uses deep neural network to approximate the action-value function. Classical Q-learning agents have some limited applicability in some domains where useful features can be handcrafted. To use reinforcement learning successfully in real-world situations, Mnih et al. [9] proposed this single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks–a central goal of general artificial intelligence. They use neural networks with several layers of nodes in order to build up progressively more abstract representations of the data and learn concepts such as object categories directly from raw sensory data. As a consequence, the deep Q-network could learn successful policies directly from high-dimensional sensory inputs. The deep learning networks have some stability problems to improve, and not using the same RMSProp definition that many deep learning libraries provide.

The theoretical foundation of DQN is less well understood. J.Fan et al. [10] provide the theoretical guarantees for Deep Q-Network in both algorithmic and statistical perspectives. The statistical error characterizes the bias and variance that arise from approximating the action-value function using deep neural network, while the algorithmic error converges to zero at a geometric rate.

Mnih et al's paper [9] not implement the DQN algorithm, and simply represent the results which often be used as a benchmark. Melrose Roderick et al. [11] implemented a Deep Q-Network(DQN) to play the Atari games and replicated the results of Mnih et al. Their implementation is also designed to be flexible to different neural net network architectures and other problem domains. They use the NVIDIA CUDA Deep Neural Network library (cuDNN) to run forward and backward passes on common neural network layers optimized specially for the NVIDIA GPUs.

3.2.2 Model Based On Double Q-Network. Since DQN Algorithm uses the same Q estimate value for selecting and evaluation in action selection. This will lead to maximization bias.[12]. So Zeng et al.[13] proposed to use Double Q-Network to select devices from all devices to improve the performance of Federated Learning by decoupling action selection and evaluation process. The double-DQN algorithm in Federated Learning mentioned by Zhang et al.[14] is based on choosing the action with the largest Q value in the next state and using the selected action to update target Q and can be expressed as:

$$Qtarget = r + \gamma Q(s', \underset{a}{\operatorname{argmax}} Q(s', a; \theta); \theta\neg) \tag{1}$$

where θ is parameter in evaluated network and $\theta-$ represents parameter in target network. $\theta-$ get updated with θ after a certain steps. After applying double-DQN. We expected to see the model to converge and achieve accuracy target faster according to the simulation result get by Nguyen et al.[15]

4 Implementation Details and Experiments Setup

In this section, we discussed in details about our implementation/changes to the code, and the experiments setup we used to evaluate our DQN implementation. We completed the project in the following steps:

First, we ran the given *flsim* code to see if the federated learning system can run as expected. However, the given code has many bugs and missing parts. For example, when the clients are sending reports to the server, the local updated weights were not included in their report. Thus we have to fix the code first. Some other bugs include various typos in code.

Second, after the Federated Learning system is set up, we would like to reproduce the Figure 3 in the paper, which compared the number of communication rounds required to reach 99% testing accuracy for FedAvg on IID and non-IID data settings. We also compared their performance with using K-means and K-clusters algorithms for device selection. We have to overcome many difficulties such as we had to understand the config file settings to set up correct config for running IID and non-IID clients. The K-means server also has a bug in it that we had to fix it.

Third, we implement the PCA model to compress the state space as stated in the paper. The original code does not provide an implementation, so we had to implement it by ourselves. We tested our implementation by plotting the Figure 4 in the paper which shows the clustered clients by projecting their weights to 2-dimensional space. With the implementation of PCA, the original approximately 410,000 parameters of all devices and global models were reduced to only about 10,000 parameters for state representative.

Fourth, we then implement the DQN model for device selection. Specifically, we implemented two servers, i.e. DQN-Train server and DQN server. The DQNTrain server is used to train the DQN model through multiple episodes of FL processes. Then once the DQN model is trained, the DQN server loaded the trained model to conduct testing in a new round of Federated Learning. We recorded the training performance of DQN using its total reward in each episode and its testing performance using testing accuracy of each round.

Fifth, we evaluated the DQN on 2 settings, i.e. selecting 10 devices out of total 100 clients, and selecting 4 devices out of total 20 clients. In the 100-client case, each client has 600 data while in the 20-client case, each client has 3000 data. We compared the trained DQN performance with Fedavg, K-Means, and k-clusters as baselines.

Sixth, after observing the bad performance of DQN which is not as we expect, we analyzed why DQN did not work well. Then we proposed a new reward function based on the improvement on testing accuracy rather than on the difference to the target accuracy. We compare the training and testing performance of two reward functions in the 100-client and 20-client settings as mentioned above.

The experiments of training and testing DQN model was done on a Linux server (64 cores Intel Xeon Silver 4313 CPUs and 3 Nvidia A30 GPUs with 24 GB each). Each round of training episode has max 50 steps (max 50 FL communication rounds), which takes about 11 mins per episode. We trained the DQN for about 200 episodes for each settings.

5 Evaluation Results and Discussions

5.1 FedAvg in Non-IID Settings

FED-AVG is a client selection strategy which selects ten clients from clients pool randomly to participate the training each round. K-Center is strategy which do k cluster classification on clients pool first and choose clients come from same cluster to participate training. K-mean is strategy to choose k clients with minimum mean, which indicate these clients are most similar to train. As shown in Figure 3, FED-AVG only takes around 60 rounds to achieve the 99% accuracy.But when comes to non-IID data, all three methods take almost triple communication rounds to achieve the target.

5.2 Clustering Clients using PCA weights

As demonstrated in above sections. We treat weights from clients as DQN training state spaces. Since it's relatively too large to train. We implement PCA to reduce the weights from clients to two dimension. As shown in Figure 4, after

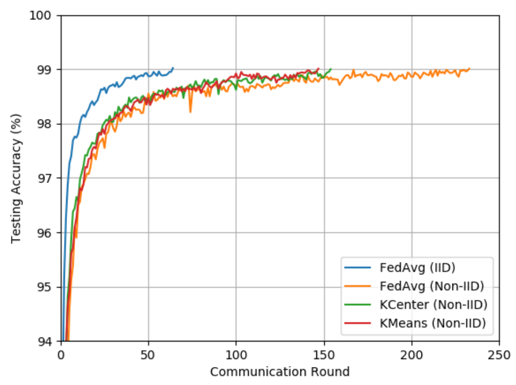
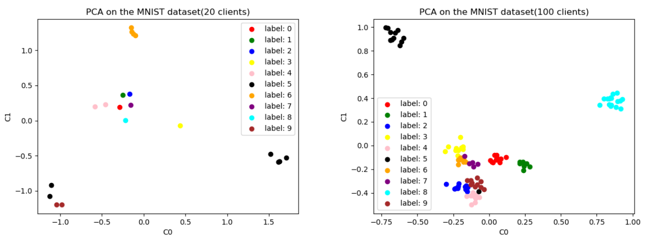


Figure 3. Training model on non-IID MNIST data.



(a) PCA for 20 clients **(b)** PCA for 100 clients

Figure 4. PCA on clients weights

dimension reduction, clients can still be classified by two dimensions weights both on 20 clients pool and 100 clients pool.

5.3 DQN Training Performance with 2 Reward Functions

The original reward function as discussed in above is

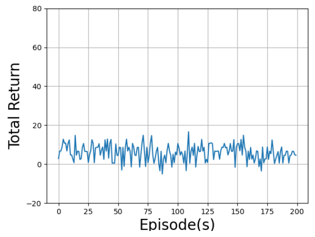
$$r_t = \Xi^{(\omega_t - \Omega)} - 1, t = 1, \ldots, T \tag{2}$$

However, we discover the original reward function can not reflect the return from training comprehensively. Because our DQN model will select one client to participate training each round. If we suppose the DQN model select the same clients every round, it will equivalent to use one client to train the global model. The global model accuracy will still increase during training. But the selection process doesn't get optimized at all and it is unreasonable. So we propose a new reward function:

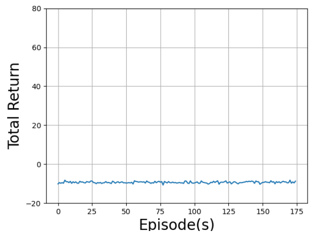
$$r_t = \begin{cases} \Xi^{(\omega_t - \omega_{t-1})} & \omega_t > \omega_{t-1} \\ -\Xi^{(\omega_{t-1} - \omega_t)} & \omega_t < \omega_{t-1} \end{cases} \quad t = 1, \ldots, T \tag{3}$$

As shown in equation (3), now reward function will provide positive reward if accuracy has improvement meanwhile give penalty if accuracy drops.

In the experiment, we perform DQN training on both reward function with two settings: picking 10 clients from 100 total clients(shown in Figure 5.) and picking 4 clients

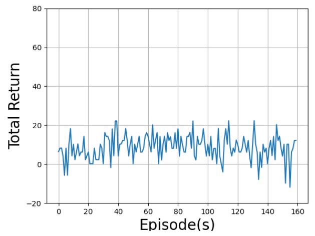


(a) 10-100 DQN training with new reward function

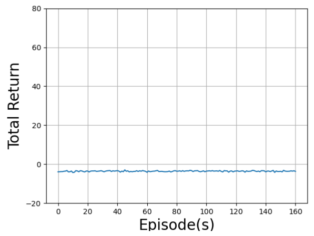


(b) 10-100 DQN training with given reward function

Figure 5. 10-100 DQN training reward



(a) 4-20 DQN training with new reward function



(b) 4-20 DQN training with given reward function

Figure 6. 4-20 DQN training reward

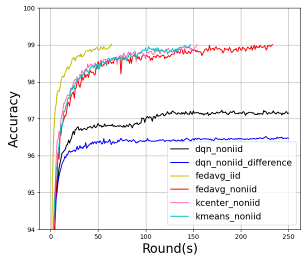
from 20 total clients(shown in Figure 6.) However, even new reward function provides more clear change on reward during training process, the result indicates DQN agent doesn't get improved during training because the reward doesn't increase as training process running for both settings.

5.4 DQN Inference Performance

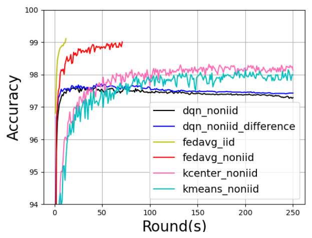
In this section we compared our DQN strategy with traditional strategies for two settings. First setting is 4-20 which represents picking 4 clients with best performance within total 20 clients. 10-100 represents picking 10 top performance clients out of 100 clients. The results shown in Figure 7.

In 4-20 setting, our DQN strategy reaches high accuracy in much fewer communication rounds than K-mean and K-center strategy. But it's not clearly prove that DQN is a better strategy because K-mean and K-center tend to underperform in small sample collections. The problem of DQN can be observed more obviously in 10-100 setting. As number of clients increase and state space increase in DQN, the performance of DQN strategy perform worse than every other strategy on both converge speed and final accuracy achieved.

In conclusion, random selecting clients for federal provides better result then selecting clients by trained DQN doesn't improve accuracy or speed in our experiment. we will discuss the potential reason for failure of DQN in the next section.



(a) 10-100 DQN compare



(b) 4-20 DQN compare

Figure 7. DQN Accuracy v.s. communication rounds compare with traditional methods

5.5 Why DQN did not work?

The major reason DQN doesn't work might because we only select a single client each round during training. So the return from each round has a huge bias. This also leads to problem that DQN agent doesn't explore enough. It is highly possible that there are many clients are never captured by agent during training. But increasing number of clients increase the cost of DQN exponentially. For example, choosing 2 clients within 100 clients will increase action space from 100 to over 4950. And due to limitation of time and device, we can not afford such huge cost.

Also the memory size to record trajectories maybe too small in our DQN model. Consider the 10,100 state space and 100 action space in setting, only 200 episodes in each trajectory is relatively small. Which may cause the training ended before agent explored the solution with best return. However, training 200 episodes takes more than 30 hrs. Increasing the number of training episodes will increase the time cost significantly, which we can not achieve in this project.

Besides, even with the new reward function we proposed, reward function cannot reflect actual return of DQN training accurately because test accuracy will still increase after each training round regardless the selection of client. For example, even the server select the same device for every communication round, the testing accuracy will still increase with more rounds (in case is like multiple rounds of stochastic gradient descent with the same data used for every round).

Last but not least. DQN is not able to provide ideal performance for problems which has strong dependency between actions such as the some Atari games (Double Dunk and Montezuma) mentioned in the class. And our client selecting problem is a typical problem of that type since the selection before and then in FL are very strongly correlated in learning result.

6 Future work

The DQN model in our current implementation did not reproduce the converging rewards as shown in the paper. Thus, in our future work, we plan to:

1. Try in a settings with small number of total clients, such as selecting 2 devices each round out of total 10 devices.
2. Try the policy gradient RL models like Actor-Critic models.
3. Improve the DQN model such as increasing replay memory size and designing more precise reward functions.

7 Conclusions

In conclusion, we successfully build the federal learning model with client choosing mechanism based on Deep Q-Learning agent. Unfortunately, our experiment on implementing DQN to optimize federal learning on non-iid data did not reproduce the work shown in the paper. Based on our analysis, we believe the reward function proposed by the original paper is problematic and DQN is not suitable for device selection process during Federated learning which has strong dependency between actions. But implementation not just DQN but all reinforcement learning method to improve performance of federal learning is still a interesting research field with a huge potential in the future.

References

[1] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1698–1707. IEEE, 2020.

[2] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[3] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 54–66, 2021.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[5] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, 2019.

[6] Hyunsung Cho, Akhil Mathur, and Fahim Kawsar. Flame: Federated learning across multi-device environments. *arXiv preprint arXiv:2202.08922*, 2022.

[7] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243*, 2020.

[8] Peter Dayan Christopher J.C.H Watkins. Q-learning. pages 279–292, 1992.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[10] Zhuoran Yang, Yuchen Xie, and Zhaoran Wang. A theoretical analysis of deep q-learning. *CoRR*, abs/1901.00137, 2019.

[11] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *CoRR*, abs/1711.07478, 2017.

[12] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[13] Rongfei Zeng, Chao Zeng, Xingwei Wang, Bo Li, and Xiaowen Chu. A comprehensive survey of incentive mechanism for federated learning. *arXiv preprint arXiv:2106.15406*, 2021.

[14] Jiaxiang Zhang, Yiming Liu, Xiaoqi Qin, and Xiaodong Xu. Energy-efficient federated learning framework for digital twin-enabled industrial internet of things. In *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1160–1166. IEEE, 2021.

[15] Huy T Nguyen, Nguyen Cong Luong, Jun Zhao, Chau Yuen, and Dusit Niyato. Resource allocation in mobility-aware federated learning networks: A deep reinforcement learning approach. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE, 2020.