FeDQN: A Federated Learning Approach for Training Reinforcement Learning Agent of Atari Games

Abdullah Akgül, Hızır Can Bayram

Istanbul Technical University, Artificial Intelligence (BLG521E) Course

***Abstract*— Atari games draw a good attention for a different range of people for years. This attention has recently turned into an AI perspective for computer scientists for various reasons. Firstly, games are simple and this leads a controlled environment. Secondly, games are not that complex, thus they can be used as benchmark for a wide variety of different agents. Reinforcement learning has gained popularity in the last decade and it is commonly used in computer games to train agents. Since reinforcement learning requires a lot of data to train a good agent, requirement for data with a great variability is more important than ever nowadays. Federated learning, on the other hand, has opened great era for machine learning agents trained in different local devices via a connected system. Every device (i.e., a client) connected to a server, trains an agent with its dataset in its environments and sends its learnt weights to the server. Afterwards, the server aggregates these weights and creates global model weights with them. These weights aggregate information and transfer knowledge from different datasets collected from different environments. Such pipeline endows the global model with a generalizability power as it implicitly benefitted from the diversity of the local datasets. In this work, we, to the best of our knowledge, proposed the first federated reinforcement learning pipeline for playing atari games.**

I. INTRODUCTION

Playing atari games has taken a big part of generation X, Y and even Z with a reference to the past. Their simple yet suitable environment provides computer scientists a controlled environment where they can do their experiments an easily and effective way. [1] noticed such convenient environment of atari games and created a toolkit with a great API to play around. Pong, one of atari games in this toolkit, is a commonly played game for years. The goal in Pong is to pass a ball behind an NPC for 21 times. Reinforcement learning [2], a rapidly growing field of machine learning, is a perfect match when it comes to training an agent for a computer game that has enough data and a good environment. [3] proposed the first deep learning model with Q-learning, DQN, that trains an agent for an atari game. Despite of its success, DQN has an important drawback; it biases how the game goes in the current environment. In order to increase the variability of the knowledge, [4], [5] proposed such methods. Federated learning, on the other hand, is an emerging field with lots of benefits such as data privacy, decentralized training, data variability and so on [6]. Its main idea is to train local models, called clients, with their own datasets and to send their updated weights to a server. This server aggregates those weights in order to have a global weights with information of all clients without really using their datasets. Federated learning has various reinforcement learning applications such as [7], [8], [9]- among others. In our setup, we trained different agents in different environments for various rounds for each experiment. In each round, the server aggregates those informations and comes up with a global model and we have a global at last. We will introduce our pipeline with details in the upcoming sections.

II. ATARI GAME: PONG

In this project, we chose Pong game from Gym Atari environment[1] for benchmarking our experiments. This game is inspired by table tennis. Pong is played with two players. The stick to the left, is an NPC in our Gym environment. The stick in the right, however, is controlled by our agent. In this project, we used a reinforcement learning, RL, agent to play with the hard-coded agent so as to beat it. Various screenshoots of the game, taken from our Gym environment, are displayed in Fig. 1.

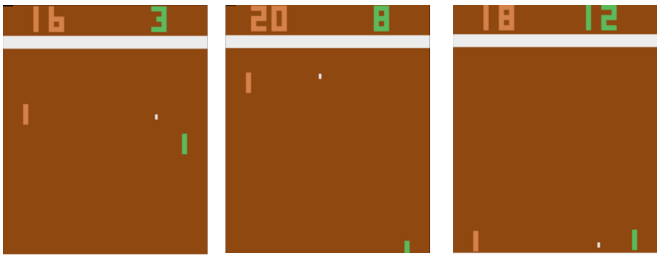

Fig. 1. Pong Game

One game is over when either of the sticks passes the ball behind the other stick and the game is over when either of the sticks beats 21 times the other. In the game, an agent can do three actions. These actions are; remaining where it is, going up and going down. In the environment, however, six actions are provided, and they are; 'NOOP', 'FIRE', 'RIGHT', 'LEFT', 'RIGHTFIRE', and 'LEFTFIRE'. 'NOOP' and 'FIRE', 'RIGHT' and 'RIGHTFIRE', 'LEFT' and 'LEFTFIRE' actions are actually the same.

In this project, we used 'PongDeterministic-v4' environment for training.

III. METHODS USED

In this work, we outline our methods as follows: A) image processing methods for feeding images to the network,

[1] https://gym.openai.com/envs/#atari

B) implementation of deep q learning, DQN, C) federation pipeline for DQN.

A. Preprocess

Gym environment provides images of size $210 \times 160 \times 3$. First off, we convert images to gray-scale format. Then, we resize them into 84×110. After that, we crop these images starting from their center in order to get rid of scoreboards in them. Finally, we interpret the numbers as integers and final size of images is 84×84, and they are ready to be fed to the network. This procedure is summarized visually in Fig. 2.

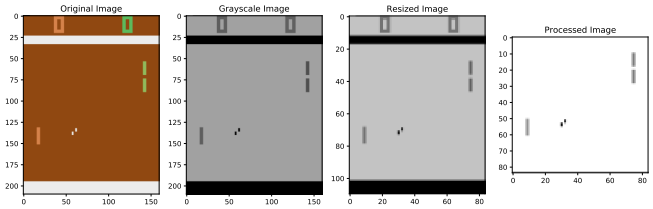

Fig. 2. Preprocess Pipeline

B. Deep Q Learning: DQN

In Q learning, agent will generate the maximum reward which is also called Q-value formulated with:

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a) \tag{1}$$

where γ is discount factor. $Q(s',a)$ depends on $Q(s'',a)$ and it is discounted again. So Q values can be interpreted as;:

$$\begin{aligned} Q(s,a) = \gamma \max_a Q(s',a) + \gamma^2 \max_a Q(s'',a) \\ + ... + \gamma^n \max_a Q(s''^{...n},a) \end{aligned} \tag{2}$$

This formula is recursive, with some assumptions this formula can be implemented as:

$$\begin{aligned} Q(S_t,A_t) = Q(S_t,A_t) + \alpha \Big(R_{t+1} \\ + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t) \Big) \end{aligned} \tag{3}$$

where α is learning rate. Finally the optimization is done with below formula:

$$\mathcal{L} = E\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s',a') \right)}^{target} - \overbrace{Q(s,a)}^{predicted} \right\|^2 \right] \tag{4}$$

Since we have discrete action space, and the game can be considered as an easy one, we decided to use the Deep Q Learning reinforcement learning method as value learning. Instead of Q-table, we used a basic neural network. Game state is not specified, thus the game state will be created with a raw image taken from the game and then it is processed. Game score is defined as reward and it is used for reinforcement signal.

1) Input Space: As stated in section A, game states, raw images, are processed and converted to 84 by 84 images. In order to recognize past activities, we used 4 stacked frames. Final input size is of $84 \times 84 \times 4$.

2) Output Space: Since the game has 6 actions, final layer of our neural network consists of 6.

3) Architecture: We formed a basic neural network. Firstly, we used convolutional layers for learning main patterns from images, then, we append linear fully connected layers to the network. Stacked images are fed to the image and values of the actions are taken as output. The architecture of the network is shown in Table I. We used 0 padding for each convolutional layer. These layers are followed by RELU activation function.

TABLE I
DQN ARCHITECTURE

Conv Name Linear Name	Channels(In×Out) In Size	(Kernel× Stride) Size Out Size
Conv1	4×32	8×4
Conv2	32×64	4×2
Conv3	64×64	3×1
Linear1	3136	512
Linear2	512	6

C. Federated Learning

In this part, we will mention federated optimization. There are N clients in the pipeline and they all sent their model weights to the server at the end of the round t. We can denote client j's loss in round t with T_j episodes as follows:

$$f_j(\mathcal{W}_j^t) = \frac{1}{|T_j|} \sum_{i=1}^{T_j} \mathcal{L}(\mathcal{T}_{ji}; \mathcal{W}_j^t) \tag{5}$$

Using the function above, we can express federated loss function of all clients in round t as follows[6]:

$$f(\mathcal{W}^t) = \sum_{j=1}^{N} f_j(\mathcal{W}_j^t) \tag{6}$$

We, on the other hand, formulate federated model's weights in the following way:

$$\mathcal{W}^{t+1} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{W}_n^t \tag{7}$$

where, $\mathcal{W}_n^t$ is weights of all layers of nth client in tth round and $\mathcal{W}^{t+1}$ is the federated weights of all layers in $t+1$th round.

IV. EXPERIMENTS AND RESULTS

In this project, we basically set up two experiments: reinforcement learning and reinforcement learning with federated learning. In federated learning experiment we used 5 clients and 0.4 fraction which means 2 clients are trained in every round. We trained the clients for 50 episodes and in each episode the game is over. Therefore the step numbers can be

vary but number of episodes are the same. We train 25 round so that the total number of episodes reaches 2500. In order to use reinforcement learning agent as a baseline, however, we trained the agent with 2500 episodes composed of 25 round and in each round 100 episodes are used for training because there is only one agent in this setup.

Starting epsilon and final epsilon are equal to 1 and 0.02, respectively. Epsilon is decreased over the steps. We used 1773 as a seed number. The length of the replay memory is set with 250000. The gamma value is used as 0.99. We used mean squared error as loss. We used RMSprop optimizer with 0.0001 learning rate.

The experiments are carried out on RTX 2080.

The train and eval results are shown in Fig. 3, 4.

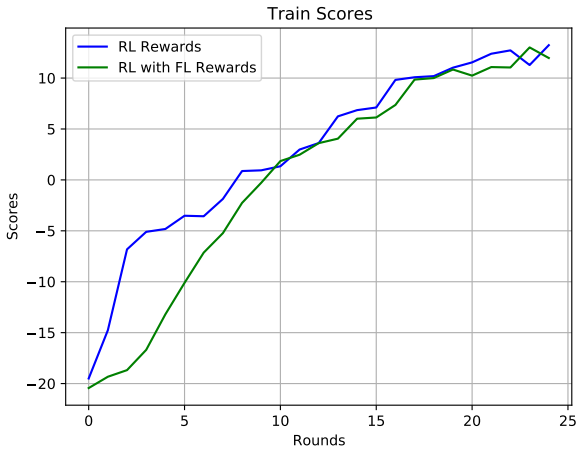

Fig. 3. Train Results

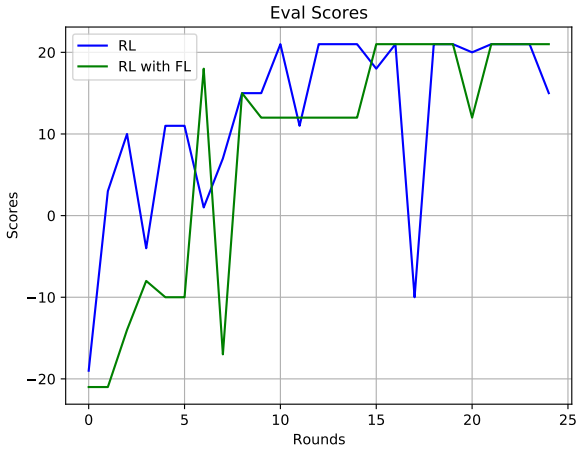

Fig. 4. Eval Results

According to train and evaluation results, we can say that both methods are able to reach the maximum score. So we can use federating learning with reinforcement learning without any concern. In addition to that, shown in Fig 5, when we use the early stopping technique just as the evaluation score reaches score of 21, the reinforcement learning method could be trained only 10 rounds which equals 1000 episodes; whereas, the federated learning method could be trained with 750 episodes, outperforming reinforcement learning agent. Another upside of FeDQN is, since parallel training is allowed, we have 25% better convergence based on what episodes those agents need to be trained.

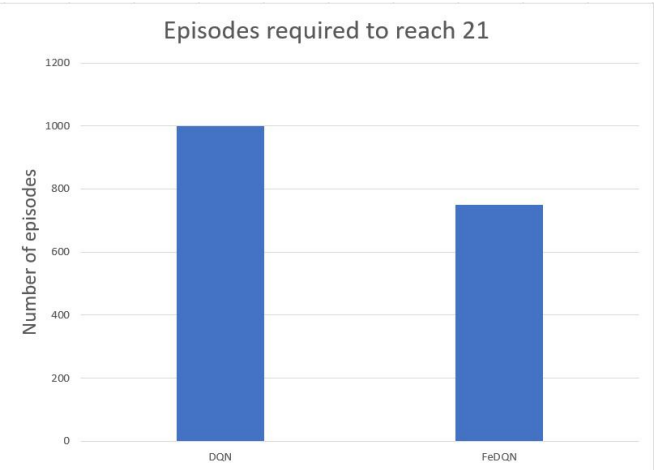

Fig. 5. Required Number of Episodes to Reach Maximum Score

V. CONCLUSIONS

In this work, we presented how we can train a reinforcement learning agent for atari games using federated optimization. We showed that clients can share their knowledge without actually sharing their datasets and environments. FeDQN strategy outperformed DQN strategy in both, being trained using less episodes and less training time. In our future work, we can extend the complexity of our environment from atari games to a complex multiplayer strategy games in order to devise knowledge for much more complex environments and more suitable pipeline for more realistic games.

REFERENCES

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International Conference on Machine Learning*, pp. 1407–1416, PMLR, 2018.

[5] K. Clary, E. Tosch, J. Foley, and D. Jensen, "Let's play again: Variability of deep reinforcement learning agents in atari environments," *arXiv preprint arXiv:1904.06312*, 2019.

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

[7] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 234–243, IEEE, 2020.

[8] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1698–1707, IEEE, 2020.

[9] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated reinforcement learning," *arXiv preprint arXiv:1901.08277*, 2019.