Federated Reinforcement Learning for Controlling Multiple Rotary Inverted Pendulums in Edge Computing Environments

Hyun-Kyo Lim
Dept. of Interdisciplinary Program in Creative Engineering
Korea University of Technology and Education
Cheon-an, Korea
glenn89@koreatech.ac.kr

Ju-Bong Kim
Dept. of Computer Science Engineering
Korea University of Technology and Education
Cheon-an, Korea
jubong1992@gmail.com

Chan-Myung Kim
Advanced Technology Research Center
Korea University of Technology and Education
Cheon-an, Korea
cmdr@koreatech.ac.kr

Gyu-Young Hwang
Dept. of Computer Science Engineering
Korea University of Technology and Education
Cheon-an, Korea
to6289@koreatech.ac.kr

Ho-bin Choi
Dept. of Computer Science Engineering
Korea University of Technology and Education
Cheon-an, Korea
chb3350@koreatech.ac.kr

Youn-Hee Han*
Dept. of Computer Science Engineering
Korea University of Technology and Education
Cheon-an, Korea
yhhan@koreatech.ac.kr

[1] ***Abstract*—Reinforcement learning has recently been studied in various fields and also used to optimally control real devices (e.g., robotic arms). In this paper, we try to allow multiple reinforcement learning agents to learn optimal control policy on their own devices of the same type but with slightly different dynamics. For such multiple devices, there is no guarantee that an agent who interacts only with one device and learns the optimal control policy will also control another device well. Therefore, we may need to apply independent reinforcement learning to each device individually, which requires time-consuming effort. To solve this problem, we propose a new federated reinforcement learning architecture where each agent working on its independent device shares their learning experience with each other, and transfers a mature policy model parameters into other agents. We incorporate the Actor-Critic PPO algorithm into each agent in the proposed collaborative architecture, and propose an efficient procedure for the gradient sharing and the model transfer. We also use edge computing to solve network problems that occur when training multiple real devices at the same time. Using multiple rotary inverted pendulum devices, we demonstrate that the proposed federated reinforcement learning scheme can effectively facilitate the learning process for multiple devices, and that the learning speed can be faster if more agents are involved.**

***Index Terms*—Actor-Critic PPO, Edge computing, Federated reinforcement learning, Multiple RIP control**

I. Introduction

Recently, reinforcement learning has been applied to various fields and shows better performance than the human. Reinforcement learning [1] is how the agent observes the environment and chooses an action that maximizes the cumulative future reward. In particular, after the development of Deep Q-Network (DQN) by Google DeepMind, reinforcement learning has been applied to Atari Games in 2015 [2]. As a result, the optimal control using reinforcement learning has been studied mostly in the area of game playing.

On the other hand, the real-world control systems, such as inverted pendulums, robot arms, quadcopters, continuum manipulators, and so on, are of different categories of nonlinearities. As one of the most effective ways to develop the nonlinear control strategy, reinforcement learning-based control schemes train an agent to learn the optimal control policy by directly interacting with such systems in a trial-and-error manner [3]–[6]. In our previous work [7], we also proved that a DQN agent can control a rotary inverted pendulum (RIP) successfully, even though the agent is located remotely from the pendulum. An imitation learning approach was applied to reduce the learning time and mitigate the instability problem in the previous study.

Multiple devices of one type are often installed and utilized simultaneously in a domain. In order to control such multiple robotic arms optimally, a reinforcement learning agent that controls one device can collaborate with other agents that control the other devices in order to learn its optimal control policy faster. Such a collaboration is able to accelerate the learning process and mitigate the instability problem for training multiple devices.

Although multiple devices of the same type are produced on the same manufacturing line, their physical dynamics are

[1]This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1A6A1A03025526).

Corresponding Author: Prof. Youn-Hee Han

usually somewhat different. Therefore, there is no guarantee that an agent who interacts only with one device and learns the optimal control policy will control another device well. A simple copy of a mature control policy model to multiple devices cannot be a solution, but a kind of cooperative method is required to train multiple devices efficiently. We consider that a cooperative reinforcement learning agents sharing learning outcomes can have more generalization capability than an agent that interacts only one device.

Bonawitz et al. [8] proposed a new machine learning system based on *federated learning*, which is a distributed deep learning approach to enable training on a large corpus of decentralized data residing in multiple devices like a mobile phone. The federated reinforcement learning enables learning to be shared among other agents on devices by conducting learning in separate environments through distributed multi-agents and collecting learning experiences through a broker. In this paper, we propose a new federated reinforcement learning scheme for controlling multiple devices of the same type in edge computing environments.

In the proposed scheme, each agent on multiple devices shares its learning experiences, i.e., a gradient of loss function computed on each device, with each other. This strategy enhances the generalization ability of each agent. Each agent also transfers its deep learning model parameters into other agents when its learning process is finished, so that other workers can accelerate its learning process by utilizing the mature parameters. Therefore, the proposed federated learning scheme consists of two phases: 1) gradient sharing phase and 2) transfer learning phase. Inside the overall procedure of the proposed scheme, the actor-critic proximal policy optimization (Actor-Critic PPO) [9]–[11] is incorporated instead of DQN. We evaluate the performance of our federated reinforcement learning scheme incorporating Actor-Critic PPO with the three RIP devices in edge computing environments.

The remainder of this paper is organized as follows. In Section II, we review related studies and state the motivation for our work. In Section III, we describe the overall system architecture and procedure of the proposed federated reinforcement learning scheme. In Section IV, we explain the operation methods of the gradient sharing and model transfer, and present the details of the Actor-Critic PPO algorithm used for reinforcement learning. In Section V, we prove the effectiveness of the proposed scheme by applying it to three real RIP devices. Finally, we provide concluding remarks and future work in Section VI.

II. RELATED WORK

Federated learning is a distributed machine learning technique that trains an algorithm across multiple decentralized edge devices holding local data samples, without exchanging their data samples. [8]. It allows multiple edge devices to jointly train a deep learning model on their local data without any of the devices revealing their data to a centralized server or other devices. This form of privacy-preserving collaborative learning is achieved by following a simple three-step procedure. In the first step, all devices download the latest master model from the server. Next, the devices improve the downloaded model by using stochastic gradient descent (SGD) algorithms based on their local training data. Finally, all devices upload their locally improved model updates back to the server, where they are gathered and aggregated to form a new master model. These steps are repeated until a certain convergence criterion is satisfied, or lasted for a long period in order to improve the deep learning model continuously.

The combination of federated learning and reinforcement learning, namely federated reinforcement learning (FRL), was first studied in [12]. In the study, the authors demonstrate that the FRL approach is capable of making full use of the joint observations (or states) from an environment, and outperforms a simple DQN with partial observation of the same environment. FRL is also applied into autonomous driving [13], where all the participant agents make steering control actions with the knowledge learned by others, even when they are acting in very different environments. Even in robot system control, FRL is used to make robot agent models fuse and transfer their experience so that they can effectively use prior knowledge and quickly adapt to new environments [14]. However, the previous FRL schemes are evaluated and verified in software such as games or simulation rather than real devices. In addition, Actor-Critic PPO has not yet been applied into the FRL research.

III. SYSTEM ARCHITECTURE & OVERALL PROCEDURE

In this section, we describe the proposed federated reinforcement learning system architecture and overall procedure to allow multiple agents to control their own devices. The proposed system consists of one chief node (in cloud server) and 3 worker nodes (in edge node). The workers have their own independent environment (i.e., device) and train its actor and critic models to control the environment optimally through its reinforcement algorithm. On the other hand, the chief node mediates the federated work across the 3 workers and ensures that each worker's learning process is synchronized.

A. System Architecture

As the real device environment, we choose the RIP device (Quanser QUBE^{TM}-Servo 2 [15]). It is a highly unstable nonlinear device and has been used as a common device in the nonlinear control engineering field. The system configuration for controlling the real multiple devices optimally is shown in Fig. 1. In the system, there are three workers and one chief, and each worker is connected with its RIP device. The RIP device is equipped with optical encoders that provide feedback on the angular position and angular velocity of the pendulum and the motor. A Raspberry Pi is connected to the RIP device and is also connected to a worker. The Raspberry Pi receives the motor power index (i.e., control input) from the worker's in edge node reinforcement learning agent, converts it into a voltage value, and finally sends it to the RIP device. It also receives the angular position and angular velocity (i.e., state

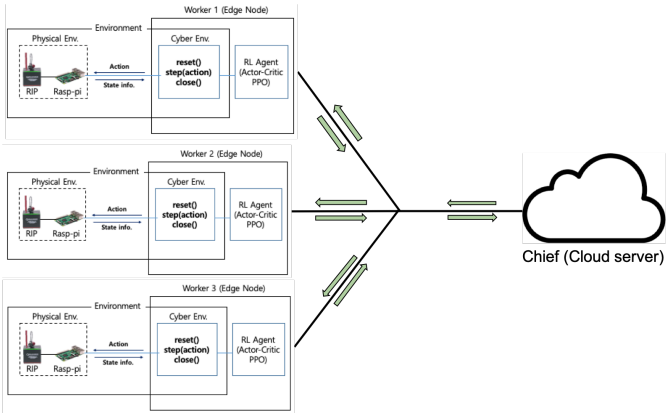

Fig. 1. The system configuration for the proposed federated reinforcement learning

information) of the pendulum and the motor from the RIP device, and sends them to the worker's reinforcement learning agent. There is also a switch for the connection between the three workers and the chief.

The reinforcement learning has been mostly studied in software environments. In such software environments, all components are located in a cyber space and implemented as a kind of software. However, as shown in Fig. 1, the real device is deployed as a tangible product in the physical space and the worker including our reinforcement learning agent is usually located in the cyber space because of the device's constrained resource. Therefore, it is important to make the two components interact in a real-time manner. In addition, it is also vital that the reinforcement learning agent should not be extensively modified to control the physical device. That is, the reinforcement learning agent should perform the control task in the same way regardless of whether the environment to be controlled exists in the cyber space or in the physical space. We configure a cyber environment in a worker and make it correspond to the physical environment, so that a reinforcement learning agent observes the state of the real device and controls it only through the cyber environment. The agent does not need to know that the real device is located in the physical space when observing and controlling it.

For the Federated reinforcement learning algorithm, a state contains four information: 1) pendulum angle, 2) pendulum angular velocity, 3) motor angle, and 4) motor angular velocity. In an episode, at every step, a new state is provided for the actor and critic models. The action is selected by each worker's actor model and the selected action is used to control the motor of the connected RIP device. The three motor power indices are -60, 0, and 60. These values correspond to the force that drives the motor. The negative sign indicates the turning direction of the motor was from right to left, while the positive sign means the turning direction of the motor was from left to right.

The reward is crucial for each worker's reinforcement learning task. The reward is determined by the cyber environment at every step. The reward value is dependent on the success or failure of each step. If the pendulum is in an upright range ±7.5 degrees (i.e., the pendulum is vertically erected) at a step, the step is successful. Otherwise, the step fails. If the step is successful, the reward value is 1 and the episode continues. If the step fails, the reward value is 0, and the episode comes to end. When an episode comes to end, a score is set to the sum of the reward values of all the steps during an episode. If the average of the reward in the last 10 episodes is 2,450, then the learning is determined to be complete.

B. Overall Procedure

Fig. 2 shows the system overall procedure for the proposed federated reinforcement learning scheme. The process where each worker sends and receives messages back to the chief is called *Round*. For a round, each worker starts a sequential interaction (i.e., an episode) with its environment at the time step t=0, and finishes at the time step T when the episode end condition is met[2]. At every time step t, the worker receives a representation s_t of the environment's state, selects an action a_t that is executed in the environment which in turn provides a reward signal r_{t+1} and a representation s_{t+1} of the successor state.

In each round, each worker's reinforcement learning algorithm calculates the gradients for the optimization of actor and critic models by using the tuples stored in the trajectory memory. For the deep learning models of a worker, the gradients are the vectors of partial derivatives with respect to the parameters of the models and they are used to find the optimal models to control the device.

The two phases, 1) gradient sharing and 2) model transfer, are synchronized and mediated by the chief. In the gradient sharing phase, the chief collects the the actor model's gradient produced by the learning process of workers (step ① in Fig. 2), averages them (step ②), and sends it back to the workers (step ③). The workers optimize the current actor model once more using the average gradient received from the chief. In our architecture, the gradient represents the experience of the learning task that a worker has made to control its device. That is, workers share their experience with each other during the gradient sharing phase.

The idea of gradient sharing was first introduced by [16]. With synchronous gradient sharing, the chief waits for all gradients to be available before computing the average and sending it back to the workers. The drawback is that some workers may be slower than others, so other workers may have to wait for them at every round. To reduce the waiting time, we could ignore the gradients from the slowest few workers (typically 10%).

After multiple rounds of the gradient sharing, a worker comes to satisfy the predefined criteria for completing the learning process. At this time, the next round is executed for the model transfer phase. During the model transfer phase, the worker completing its learning process sends its mature actor model parameters to the

[2]The episode end conditions vary according to the type of control devices and the control objective

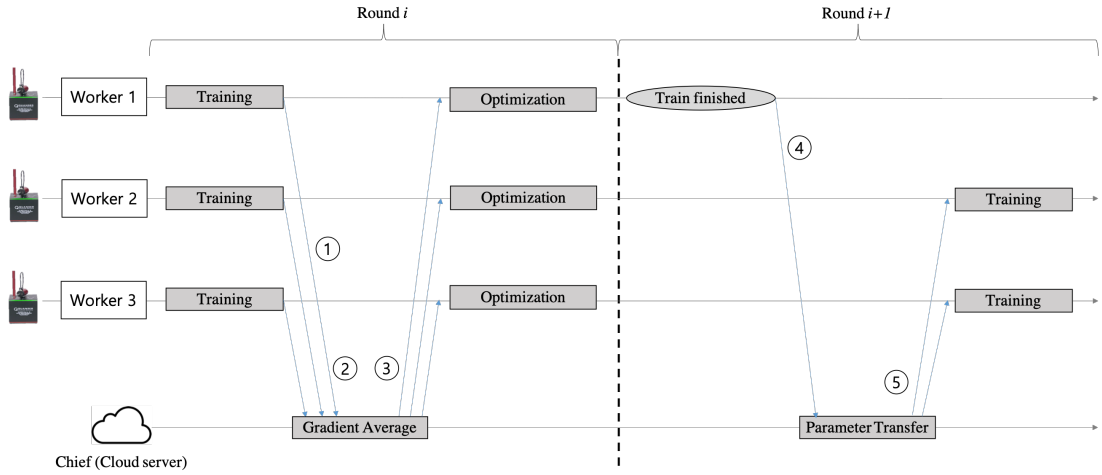

Fig. 2. The federated reinforcement learning overall procedure

(step ④ in Fig. 2). When receiving the mature actor model parameters from a worker, the chief considers them the best ones, and it transfers them to the rest another workers (step ⑤). And then, the another workers replace their model parameters with the mature model parameters, so that the leaning time can be reduced for the workers with slow learning.

Although the mature actor model is incorporated to the other workers, several gradient sharing phases may be still needed among the workers, since the inherent dynamics of each device are slightly different from each other. Therefore, the same procedure of the steps from ① to ⑤ in Fig. 2 is performed repeatedly between the rest of the workers. By continuously performing this procedure, the learning processes of all workers are completed when the predefined condition is satisfied at the last worker.

IV. Federated Reinforcement Learning Algorithm

In our study, each worker conducts individual training with the Actor-Critic PPO algorithm on an independent device. The key advantage of Actor-Critic PPO is that a new update of the policy model does not change it too much from the previous policy. It leads to less variance in learning, but ensures smoother policy update and also ensure that the worker does not go down an unrecoverable path of taking senseless actions. This feature is particularly important for optimal device control because optimal control learning for devices in the physical environment takes more time than software in the cyber environment.

An actor model (i.e., policy model) π_θ has its own model parameters θ. With an actor model, a worker performs the task of learning what action to take under a particular observed state of the device. The worker sends the action predicted by the actor model to the device and observes what happens in the device. If something positive happens as a result of the action, then a positive response is sent back in the form of a reward. If a negative occurs due to the taken action, then the worker gets a negative reward. This reward is taken in by the critic model V_μ with its model parameter μ. The role of the critic model inside a worker is to learn to evaluate if the action taken by the actor model led the device to be in a better state or not, and the critic model's feedback is used to the actor model optimization.

In a general policy gradient reinforcement learning, the objective function L^P is as follows:

$$L^P(\theta) = \widehat{E}\left[\log \pi_\theta\left(a_t|s_t\right)\widehat{A}_t\right] \tag{1}$$

where $\widehat{E}[...]$ is the empirical average over a finite batch of samples, and $\widehat{A}_t$ is an estimator of the advantage function at time step t. With the discount factor $\gamma \in [0,1]$, we use the generalized advantage estimator (GAE) [17] to calculate $\widehat{A}_t$. The GAE is

$$\widehat{A}_t = \delta_t + (\gamma\lambda)\,\delta^V_{t+1} + (\gamma\lambda)^2\,\delta^V_{t+2}\cdots(\gamma\lambda)^{U-t+1}\,\delta^V_{U-1} \tag{2}$$

where λ is the GAE parameter ($\lambda \in [0,1]$), U is the sampled mini-batch size, and $\delta_t = r_t + \gamma V_\mu(s_{t+1}) - V_\mu(s_t)$. The objective (i.e., loss) function L^V is as follows:

$$L^V(\mu) = \widehat{E}\left[L^V_t(\mu)\right] = \widehat{E}\left[\left|\hat{V}^{target}_\mu(s_t) - V_\mu(s_t)\right|\right] \tag{3}$$

where the target value of time-difference error (TD-Error) $\hat{V}^{target}_\mu(s_t) = r_{t+1} + \gamma V_\mu(s_{t+1})$. The parameters of V_μ are updated by an SGD algorithm (i.e., Adam [18]) with the gradient ∇L^V:

$$\mu = \mu - \eta_\mu \nabla L^V(\mu) \tag{4}$$

where η_μ is the learning rate for the critic model optimization.

In the actor model of PPO, instead of the objective function presented in Equation (1), the worker uses the importance sampling to obtain the expectation of samples gathered from an old policy $\pi_{\theta_{old}}$ under the new policy π_θ we want to refine. PPO inherits the benefit from TRPO, but it is much simpler to implement, allows multiple optimization iterations, and empirically presents a better sample efficiency than TRPO.

Algorithm 1: Federated RL (Chief)

for $i = 1, 2, 3, ..., M$ **do**
  $P = []$
  for $w \in W$ **do**
    Receive a message m_w from the worker w
    Append m_w into P
  end
  if *there is a message* $m_w \in P$ *s.t.* m_w *has the actor model parameter* θ_w *of a worker* w **then**
    $\bar{\theta} = \theta_w$
    Send to the workers in $W - \{w\}$ the message m_c including the model parameter $\bar{\theta}$
    $W = W - \{w\}$
  else
    collect the gradients $g_\theta^1, g_\theta^2, ..., g_\theta^{|W|}$ from all m_w in P
    $\bar{g} = \frac{1}{|W|} \sum_{t=1}^{|W|} g_\theta^t$
    Send to all the workers in W the message m_c including the average gradient $\bar{g}$
  end
  if $W == \{\}$ **then**
    Break
  end
end

With the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the PPO objective function L^{CLIP} is given by

$$\begin{aligned} L^{CLIP}(\theta) &= \widehat{E}\left[L_t^{CLIP}(\mu)\right] \\ &= \widehat{E}\left[min\big(r_t(\theta), clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\big)\widehat{A}_t\right] \end{aligned} \tag{5}$$

where ϵ is the clipping parameter. The clipped objective function L^{CLIP} does not makes PPO greedy in favoring actions with positive advantage, and much quick to avoid actions with a negative advantage function from a mini-batch of samples. The parameters of π_θ are updated by an SGD algorithm with the gradient ∇L^{CLIP} for the negative of the clipped objective function (i.e., $-L^{CLIP}$):

$$\theta = \theta - \eta_\theta \nabla L^{CLIP}(\theta) \tag{6}$$

where η_θ is the learning rate for the actor model optimization.

With Actor-Critic PPO, our federated reinforcement learning algorithm is provided in Algorithms 1 and 2. The chief and all workers share the parameter M which indicates the maximum number of rounds (i.e., episodes). The chief maintains the set of all workers W. Whenever a chief receives the actor model parameter θ_w from a worker w, the chief removes it from W at the end of round. If W is empty, the chief finishes its task. In a worker, K is the number of model optimizations in one round. The multiple model optimizations are conducted to further improve sample efficiency.

Algorithm 2: Federated RL (Worker w)

for $i = 1, 2, 3, ..., M$ **do**
  for *each step* t *of an episode* **do**
    Run the actor model π_θ for s_t and do action a_t
    Get r_{t+1}, s_{t+1} from the environment
    Store ¡$s_t, a_t, r_{t+1}, s_{t+1}$¿ into the trajectory memory
  end
  if *learning process is finished* **then**
    Send to the chief a message m_w including the actor model parameter θ_w
    Break
  else
    Update $\pi_{\theta_{old}} \leftarrow \pi_\theta$
    for $j = 1, 2, 3, ..., K$ **do**
      Get a mini-batch B from the trajectory memory (the size of B is U)
      Compute $L_t^V(\mu)$ and $L_t^{CLIP}(\theta)$ for the mini-batch B
      Compute the gradient $g_\mu = \nabla L^V(\mu)$ for $\frac{1}{U}\sum_{t=1}^{U} L_t^V(\mu)$ w.r.t. the critic model parameter μ
      Update V_μ with g_μ through SGD
      Compute the gradient $g_\theta = \nabla L^{CLIP}(\theta)$ for $\frac{1}{U}\sum_{t=1}^{U} L_t^{CLIP}(\theta)$ w.r.t the actor model parameter π
      Update π_θ with g_θ through SGD
    end
    Send to the chief a message m_w including the last gradient g_θ
  end
  Wait for a message m_c from the chief if it is not available
  if m_c *has the actor model parameter* $\bar{\theta}$ **then**
    Replace the current actor model parameter with the received $\bar{\theta}$
  else if m_c *has the average gradient* $\bar{g}$ **then**
    Update π_θ with the received $\bar{g}$ through SGD
end

V. Performance Evaluation

In this section, we apply the proposed federated reinforcement learning scheme to the real devices. We validate the effectiveness of gradient sharing and transfer learning for our federated reinforcement learning in the real devices. We also validate the effect of the number of workers on the performance of the proposed scheme.

The workers and the chief are deployed on Ubuntu 16.04 LTS. The proposed scheme using the Actor-Critic PPO algorithm is implemented using Python 3.6 and Pytorch 1.2. A multi-layer perceptron with the three hidden layers and two separate output layers is used to constitute the actor and critic models. The two models share the three hidden layers, where each layer includes 128 neurons. For the actor model, the first

output layer yields three values (i.e., three types of actions) that sum to one. For the critic model, the second output layer yields a single value to evaluate the action selected by the actor model. The other hyper-parameters of the Actor-Critic PPO algorithm are listed in Table I.

TABLE I
HYPER-PARAMETER CONFIGURATION FOR ACTOR-CRITIC PPO

Hyper-parameter	Value
Clipping parameter (ϵ)	0.9
Model optimization algorithm	Adam
GAE parameter (λ)	0.99
Learning rate for the critic model (η_μ)	0.001
Learning rate for the actor model (η_θ)	0.001
Trajectory memory size	200
Batch size (U)	64
Number of model optimizations in one round (K)	10

Fig. 3 shows the efficiency of the federated reinforcement learning according to the number of workers. For single workers and three workers, the experiments are conducted three times and the average score per episode is shown in the figure. As shown in the figure, the higher the number of workers, the faster the learning process. A single worker was completed at 816 episodes. However, three workers were completed at 319 episodes. In the proposed scheme, each worker share its learning experiences, i.e., the gradient of loss function computed on each device, with each other. Each agent also transfers its actor model parameters into other agents when its learning process is finished. As the number of workers increases, the effects of such two strategies increase.

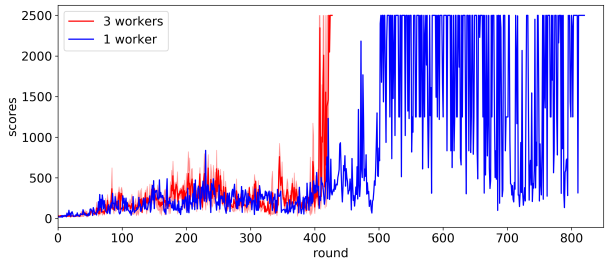

Fig. 3. Comparison of learning speed in terms of the number of workers

VI. CONCLUSION

In this work, we have shown that the proposed federated reinforcement learning scheme in edge computing environments can successfully control multiple real devices of the same type but with slightly different dynamics. We adopted Actor-Critic PPO as reinforcement learning algorithm and applied it to the federated learning architecture. The proposed approach includes the gradient sharing and model transfer methods to facilitate the learning process, and it turned out that they can expedite the learning process. We also have shown that learning is further accelerated by increasing the number of workers.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-Level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[3] W. Linglin, L. Yongxin, and Z. Xiaoke, "Design of Reinforce Learning Control Algorithm and Verified in Inverted Pendulum," in *Proceedings of The 34th Chinese Control Conference (CCC)*, July 2015, pp. 3164–3168.

[4] M. Chen, H. Lam, Q. Shi, and B. Xiao, "Reinforcement Learning-based Control of Nonlinear Systems using Lyapunov Stability Concept and Fuzzy Reward Scheme," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2019.

[5] G. Puriel-Gil, W. Yu, and H. Sossa, "Reinforcement Learning Compensation based PD Control for Inverted Pendulum," in *Proceedings of The 15th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, Sep. 2018, pp. 1–6.

[6] K. Kersandt, G. Muñoz, and C. Barrado, "Self-training by Reinforcement Learning for Full-autonomous Drones of the Future," in *Proceedings of The IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, Sep. 2018, pp. 1–10.

[7] J. Kim, H. Lim, C. Kim, M. Kim, Y. Hong, and Y. Han, "Imitation Reinforcement Learning-Based Remote Rotary Inverted Pendulum Control in OpenFlow Network," *IEEE Access*, vol. 7, pp. 36 682–36 690, 2019.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. M. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," vol. abs/1902.01046, 2019.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[10] V. Konda, "Actor-Critic Algorithms," Ph.D. dissertation, Cambridge, MA, USA, 2002.

[11] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 6382–6393.

[12] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated Reinforcement Learning," *CoRR*, vol. abs/1901.08277, 2019. [Online]. Available: http://arxiv.org/abs/1901.08277

[13] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated Transfer Reinforcement Learning for Autonomous Driving," *ArXiv*, vol. abs/1910.06001, 2019.

[14] B. Liu, L. Wang, and M. Liu, "Lifelong Federated Reinforcement Learning: A Learning Architecture for Navigation in Cloud Robotic Systems," *IEEE Robotics and Automation Letters*, vol. 4, pp. 4555–4562, 2019.

[15] "Qube - servo 2." [Online]. Available: https://www.quanser.com/products/qube-servo-2/

[16] N. Strom, "Scalable Distributed DNN Training using Commodity GPU Cloud Computing," in *Proceedings of INTERSPEECH*. ISCA, 2015, pp. 1488–1492.

[17] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *CoRR*, vol. abs/1506.02438, 2015.

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of The 3rd International Conference for Learning Representations, San Diego*, 2015.