IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 29, 2021, accepted May 18, 2021, date of publication May 24, 2021, date of current version June 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3083087

Federated Reinforcement Learning Acceleration Method for Precise Control of Multiple Devices

HYUN-KYO LIM[1], JU-BONG KIM[2], IHSAN ULLAH[3], JOO-SEONG HEO[1], AND YOUN-HEE HAN[2], (Member, IEEE)

[1]Department of Interdisciplinary Program in Creative Engineering, Korea University of Technology and Education, Cheonan 31253, South Korea
[2]Department of Computer Science Engineering, Korea University of Technology and Education, Cheonan 31253, South Korea
[3]Advanced Technology Research Center, Korea University of Technology and Education, Cheonan 31253, South Korea

Corresponding author: Youn-Hee Han (yhhan@koreatech.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Education under Grant 2018R1A6A1A03025526 and Grant NRF- 2020R1A6A3A13073735.

ABSTRACT Nowadays, Reinforcement Learning (RL) is applied to various real-world tasks and attracts much attention in the fields of games, robotics, and autonomous driving. It is very challenging and devices overwhelming to directly apply RL to real-world environments. Due to the reality gap simulated environment does not match perfectly to the real-world scenario and additional learning cannot be performed. Therefore, an efficient approach is required for RL to find an optimal control policy and get better learning efficacy. In this paper, we propose federated reinforcement learning based on multi agent environment which applying a new federation policy. The new federation policy allows multi agents to perform learning and share their learning experiences with each other e.g., gradient and model parameters to increase their learning level. The Actor-Critic PPO algorithm is used with four types of RL simulation environments, OpenAI Gym's CartPole, MoutainCar, Acrobot, and Pendulum. In addition, we did real experiments with multiple Rotary Inverted Pendulum (RIP) to evaluate and compare the learning efficiency of the proposed scheme with both environments.

INDEX TERMS Federated reinforcement learning, multi-agent, transfer learning, gradient sharing.

I. INTRODUCTION

Recently, reinforcement learning has been applied to games, robotics, and autonomous driving which required precise control and accurate results [1]–[5]. In the real-world and simulation environment RL has gained much popularity to solved complex problems of several domains. However, in robotics and autonomous driving, the result and accuracy of the RL are still at the research level and much improvement is needed to be applied in real-world scenario. Nevertheless, it is challenging to directly apply RL to the real world-world and infer accurate results. So, we need an efficient approach to solve the problems and limitations of RL for directly applying to the real world or devices.

In real-world, there are several environments which contains trainable distributed devices, such as IoT smart factories, and distributed computing environments. In these environments, multi-agent reinforcement learning is needed to control distributed devices simultaneously and precisely without human intervention. However, it is difficult to directly apply reinforcement learning to the real environment. Performing reinforcement learning on a real device runs out of computing resources or consumes parts of devices for learning. To solve this problem, it is common to first perform training in a simulation environment and then apply the trained model to the real environment and tuning the system. The main drawback of this approach is that the simulation environment, and the real environment do not match perfectly and mostly failed in the tuning process.

Reinforcement learning is being applied in several multi agent environments to investigate the interaction and share the learning between the agents [6]. In IoT environments, multi-agent-based reinforcement learning [7], [8] is being studied, which is better than the single agent-based reinforcement learning [9]. Such as Federated learning technique where many agents share learning with each other and collaboratively train a model under the control of the central server while keeping data decentralized. In particular, the federation method is suitable for training multiple devices that working in the environment and have the same dynamic

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

characteristics and objectives. In 2015, Google proposed the federated learning approach in which multi agents undertake distributed learning and send the trained model parameters to the cloud to share their learning experiences [10]. The federation policy does not share the local training data but sends the learning parameters to the cloud or center place. Federation policy approaches reduced the data transmission on the network and also solved the privacy problem. Also, reinforcement learning algorithms applying the federation policy to real devices are being studied [11]–[14]. We are motivated by previous research trends to apply federated multi-agent reinforcement learning to multiple real devices and improve learning performance. In our previous research [15]–[17], we have applied the Deep Q Network (DQN) to the RIP system and IoT devices in the Software-Defined Network environments for automatic and training control policies.

In this article, we extend our previous research using federated reinforcement learning to enable precise control simultaneously on multiple RIP systems environments. We propose an efficient federation policy by using gradient sharing and transfer learning methods for multi-agent system. Our scheme adopts Actor-Critic PPO algorithm with four types of RL environments, OpenAI Gym's CartPole, MoutainCar, Acrobot, and Pendulum for simulation, and multiple rotary inverted pendulum (RIP) real device system for real experiment. The behavior of RIP is unstable and has nonlinear characteristics, hence, it has been used widely as a testbed for nonlinear control systems. Federated reinforcement learning is based on multi-agents using gradient sharing method for exchanging the gradients calculated by all agents during learning process. In transfer learning, when one agent completes learning, it transfers the parameters of the learned model to the other agents. Based on the proposed federation policy all agents shared their learning experience (e.g., gradient and model parameters) to update the learning level. The proposed transfer learning approach does not transfer the fully trained model to the current model but transfers the model which guaranteed by the current learning model. The guarantee is assigned to the model by calculating the weights according to the learning level of the current agent. In other words, by considering the learning level of each agent during and after learning, the stability of learning is assured, and it is completed more rapidly. For the simulation and real device experiment we used Actor-Critic Proximal Policy Optimization (Actor-Critic PPO) [18]–[20],which the best performance for an agent-based reinforcement learning algorithm among other policy gradient methods. It includes Trust Region Policy Optimization (TRPO) [21], which exhibits low computation and high performance. The main contributions of the paper are summarized as follows:

1) We propose an extended federated reinforcement learning approach to allow multi-agent for controlling the simulation and a RIP system.
2) The proposed approach can accelerate the overall learning process for control policy through simulation as well as in real devices.
3) Using evolved Gradient Sharing and Transfer Learning, the proposed scheme reduce the learning time than our previous works [15]–[17].
4) In addition, the proposed scheme based on Federated Reinforcement Learning achieves the desired goal with a little learning in the presence of much noise.

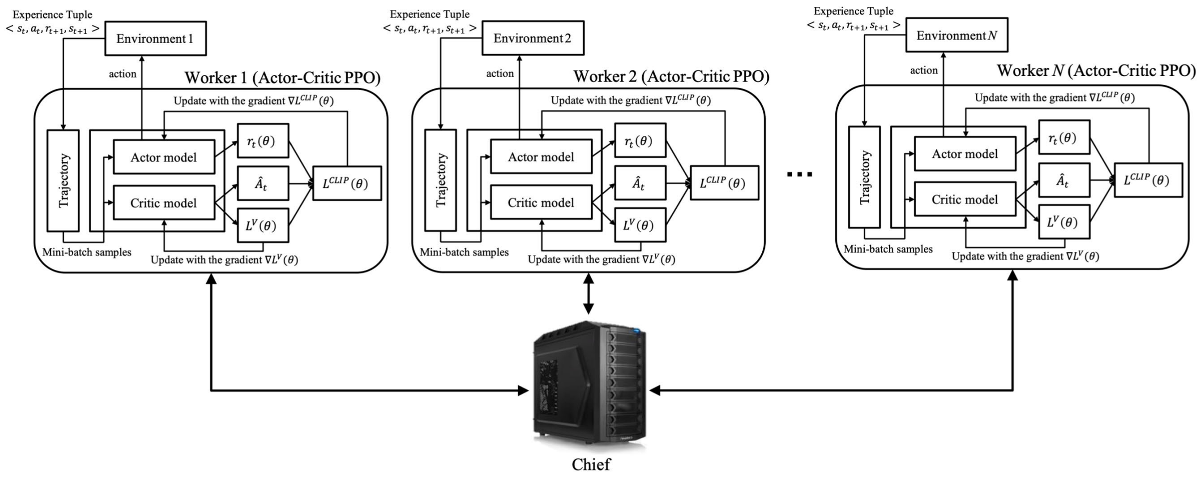
The remainder of this paper is organized as follows. In Section II, the related work and state the motivation for the proposed scheme are described. The system architecture, and details of the Actor-Critic PPO algorithm used for the proposed scheme are explained in Section III. In Section IV, the federation policy of the weight-based gradient sharing and transfer learning are described. Section V discusses the experimental result and effectiveness of the proposed scheme by applying it to OpenAI Gym and three real RIP systems, and the conclusion is made in Section VI.

II. RELATED WORK

Federated reinforcement learning is a type of multi-agent reinforcement learning [22] which is used for distributed agents system, such as games, robotics systems, and autonomous driving [2]–[5]. In [6] deep reinforcement learning is used for multi-agent cooperation and competition. Also, multi-agent reinforcement learning applied to the federation policy [23] which recently got much attraction. Subsequently, several research has been conducted that applied the federation policy to deep learning and reinforcement learning.

Bonawitz *et al.* [24] described the federation process and designed a federation system protocol for the federation policy in 2019. The federation policy performs training using local data in each distributed device and sends the calculated gradient or training model (i.e., weights and bias) to the central server. The central server processes the parameters received from each agent and returns them to the agents. Using the parameters received from the central server, the agent updates its own training model. Zhuo *et al.* [25] proposed a new reinforcement learning algorithm that builds a Q-Network for each agent using a federation policy. Each agent calculates a Q-value using its own Q-Network using local data. Each local data state is used as a global observation value; it is then combined with all the Q-values of each agent and is used as the input of the new central Q-Network. In the global Q-Network, the machine is controlled by selecting the action that maximizes the Q-value.

In addition, recent studies have also focused on transfer learning and reinforcement [26]–[29]. Transfer learning is widely used in deep learning and is a machine learning technique that reuses a trained model in a data-rich field to build a model in a field where training data is insufficient [30]. Therefore, if transfer learning is used, learning is accelerated, and the performance of the model is improved. Glatt *et al.* [28] proposed a method of applying transfer learning to reinforcement learning. They created a deep learning model with DQN to play several Atari games. To train other Atari games, the pre-trained DQN models are



Experience Tuple
$< s_t, a_t, r_{t+1}, s_{t+1} >$
Environment 1
Environment 2
Environment N
action
Worker 1 (Actor-Critic PPO)
Worker 2 (Actor-Critic PPO)
Worker N (Actor-Critic PPO)
Update with the gradient $\nabla L^{CLIP}(\theta)$
Trajectory
Actor model
Critic model
$r_t(\theta)$
$\hat{A}_t$
$L^V(\theta)$
$L^{CLIP}(\theta)$
Mini-batch samples
Update with the gradient $\nabla L^V(\theta)$
Chief

FIGURE 1. System architecture and the worker's actor-critic PPO algorithm.

transferred to new DQN models. After receiving the new DQN deep learning model, even with little training, it is faster and more accurate than training from the start. By applying transfer learning to reinforcement learning, the learning process is considerably accelerated. Vrancx *et al.* [29] proposed coordinating Q-learning (CQ-learning) to improve the learning speed and to generalize models among distributed multi-agents by utilizing transfer learning. In CQ-learning, each agent first trains in a simple and similar environment and sends the trained model to all other agents before learning in a real target environment. By this method, each agent is somewhat generalized and becomes a trained model, thus requiring little training for adapting the new environment. Therefore, the learning speed is high, the model becomes general, and it is possible to cope with dangerous situations between agents.

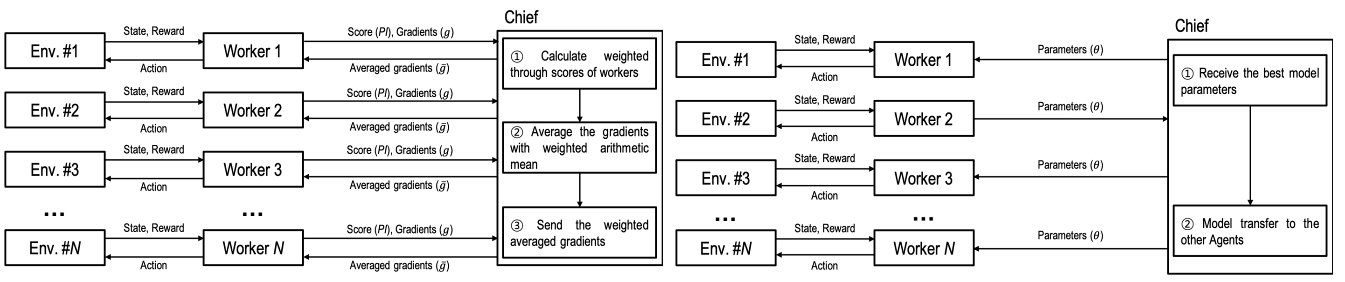
Based on the literature discussed above, and our previous research [15]–[17], we are motivated to extend our research. In the previous work [16], Gradient Sharing and Transfer Learning methods were proposed with a new federation policy for reinforcement learning. In the previous work, gradient sharing method simply collects the gradient of each worker from the chief, calculates the average, and sends it back to the workers. Also, transfer learning method simply transfers the model parameters to other workers through the chief, when the learning of a worker is completed. However, if the real device environment independently owned by workers is the same, the dynamic characteristics may be slightly different. For example, if we command the RIP device to move left by 10, it can move differently for each device, because the federation policy proposed in the previous work performs Gradient Sharing and Transfer Learning without considering the independent environment state of each worker. Hence, we extend our previous work for independent environment state of each worker, and proposed a new federation policy based on previous work.

In this paper, we present a weight-based Gradient Sharing and weight-based Transfer Learning method according to the current learning state of each worker independently. The reinforcement learning model of each worker depends on their own environment, if they train well, their weight value becomes high. Thus, gradient sharing is performed with a weight-based average, and transfer Learning performs soft transfer Learning according to the weight in the chief. The experimental result shows that the convergence rate of the proposed scheme is faster with less number of learning than the previous work. Hence, the weight-based federation policy gives better improvement and performance compared to our previous scheme [16].

III. SYSTEM ARCHITECTURE AND ACTOR-CRITIC PPO

In this section, we describe the reinforcement-learning algorithm and the proposed overall system architecture in which each worker performs learning by interacting with the environment. The proposed system is a distributed reinforcement learning system. As shown in Figure 1, the proposed federated reinforcement learning system architecture consists of the N workers and one chief. The workers have the Actor-Critic PPO algorithm, an independent environment, and perform repetitive learning. The independent environment can be a software-configured simulation or a real device. The chief mediates the federation of N workers and synchronizes their learning processes.

Actor-Critic PPO algorithm in workers has been known as a reinforcement learning algorithm for good learning performance [18], [31]. The workers undertake reinforcement learning by interacting with their independent environments. The Actor-Critic PPO is a type of policy gradient algorithm for directly learning action probabilities. Using actor-critic PPO applied with the value function is the key to increasing stability because the direct method of learning behavior probabilities is unstable. The Actor-Critic algorithm uses two networks: the Actor network and the Critic network. The Actor network determines the action when the state is determined by the environment, and the Critic network determines the value of the state.


Chief
Env. #1
Env. #2
Env. #3
Env. #N
Worker 1
Worker 2
Worker 3
Worker N
State, Reward
Action
Score (PI), Gradients (g)
Averaged gradients (ḡ)
① Calculate weighted through scores of workers
② Average the gradients with weighted arithmetic mean
③ Send the weighted averaged gradients
Parameters (θ)
① Receive the best model parameters
② Model transfer to the other Agents

(a) Proposed gradient sharing for federated reinforcement learning

(b) Proposed transfer model to the other agents for federated reinforcement learning

FIGURE 2. **The proposed federated reinforcement learning procedure for acceleration.**

The Actor has its own network parameter θ. The workers use the Actor network to perform the task of learning and actions to be taken under the environment's specific observation state. The workers send the action determined by the Actor network and observe the next state of the environment. Consequent to action, the worker receives a positive or negative reward. The reward obtained for action is considered as a network parameter by the Critic. The Critic associated with a worker learns to assess whether the actions determined by the actor have led the environment to a more positive state, and feedback from the Critic is used to optimize the Actor.

In Figure 1, the RL agents demonstrate a brief overview of the Actor-Critic-based PPO algorithm. First, Actor-Critic PPO algorithm stores the experience tuples gained from interacting with the environment in the trajectory memory and imports them as sequential finite mini-batch samples.

In traditional policy gradient algorithms, 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 $\widehat{E}[\ldots]$ is the empirical average over a finite samples (i.e., mini-batch), and $\widehat{A}_t$ is the advantage function at time step t. And we use the generalized advantage estimator (GAE) [32] 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, $\gamma \in [0, 1]$ is the discount factor, and $\delta_t = r_t + \gamma V_\mu(s_{t+1}) - V_\mu(s_t)$. The objective function L^V is as follows:

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

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 Adam optimizer algorithm with the gradients ∇L^V:

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

where η_μ is the learning rate for the critic optimization.

In the Actor of TRPO [21] and PPO, RL agent used the objective function presented in Equation (1). The RL agent in 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 that is to be refined. They maximize the following surrogate objective function L^{CPI}:

$$L^{CPI}(\theta) = \widehat{E}\left[\frac{\pi_\theta\left(a_t|s_t\right)}{\pi_{\theta_{old}}\left(a_t|s_t\right)}\widehat{A}_t\right]. \tag{5}$$

With a small value δ, the TRPO optimizes L^{CPI} subject to the constraint

$$\widehat{E}\left[KL\left[\pi_{\theta_{old}}\left(\cdot|s_t\right), \pi_\theta\left(\cdot|s_t\right)\right]\right] \leq \delta$$

on the extent of the policy update. KL indicates the Kullback–Leibler divergence (KL divergence) [33]. PPO, which is derived from TRPO, is simple to implement and requires fewer computations because it does not use KL divergence. 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

$$\begin{aligned} L^{CLIP}(\theta) &= \widehat{E}\left[L^{CLIP}_t(\mu)\right] \\ &= \widehat{E}\left[min\big(r_t(\theta), clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\big)\widehat{A}_t\right] \end{aligned} \tag{6}$$

where ϵ is the clipping parameter. The clipped objective function L^{CLIP} reduces exploratory activity to take preferred actions to gain the positive benefits of PPO. The parameters of π_θ are updated by an optimizer algorithm with the gradient ∇L^{CLIP} for the negative of the clipped objective function (i.e., $-L^{CLIP}$):

$$\theta = \theta - \eta_\theta \nabla L^{CLIP}(\theta) \tag{7}$$

where η_θ is the learning rate for the Actor optimization. Actor-Critic PPO improves efficiency by optimizing multiple model through repetitive learning.

IV. FEDERATED REINFORCEMENT LEARNING FOR ACCELERATION

In this section, we explain ways to accelerate the performance (accuracy, robust) and learning speed of federated reinforcement learning proposed in our previous research [15]–[17]. In the previous study, to perform federated reinforcement learning, we divided it into two federation policies: Gradient

Algorithm 1 Federated RL (Chief)

```
for $i = 1, 2, 3, \ldots, M$ do
    $P = []$
    for $n \in N$ do
        Receive a message $m_n$ from the worker $n$
        Append $m_n$ into $P$
    end for
    if there is a message $m_n \in P$ s.t. $m_n$ has the actor
    model parameter $\theta_n$ of a worker $n$ then
        Best parameters is $\bar{\theta} = \theta_n$
        Send to the workers in $N - \{n\}$ the message $m_c$
        including the model parameter $\bar{\theta}$
        $N = W - \{n\}$
    else
        Collect the gradients and Performance Index
        $(g_\theta^{|N|}, PI_\theta^{|N|})$ from all $m_n \in P$
        Calculate $w^n$ for gradient sharing by using
        Equation (8)
        Compute weighted average $\bar{g} = \sum_{n=1}^{N} w^n g_\theta^n$
        Send to all the workers in $N$ the message $m_c$
        including the average gradient $\bar{g}$
    end if
    if $N$ is empty then
        Break
    end if
end for
```

Sharing and Transfer Learning. Gradient Sharing accelerates learning by sharing the gradients of the agent as the learning progresses. Transfer Learning in federated reinforcement learning exchanges matures network models (satisfying terminal conditions and ending learning) to prompt other workers to complete their learning quickly. There are two differences between the federation policy of federated reinforcement learning proposed in this paper and the previous study.

Figure 2 shows the two different federation policy procedures for the proposed federated reinforcement learning scheme. Each worker initiates a sequential interaction (i.e., an episode) with the environment at time step $t = 0$ and terminates at the time step T when the conditions for terminating an episode are met. At every time step t, the worker receives a current state s_t from the environment and selects an action a_t to apply it to the environment. The selected action a_t is applied to the environment and the worker receives a reward r_{t+1} and the next state s_{t+1}. For every time step t, the worker stores the experience tuple $< s_t, a_t, r_{t+1}, s_{t+1} >$ into its trajectory memory. The size of trajectory memory is limited, and if it exceeded the size, the initially stored experience tuple is sequentially deleted.

In each episode, each worker's Actor-Critic PPO calculates the gradients for the optimization of the Actor and Critic models and each worker calculates the Performance Index (PI)that represents the performance of the current reinforcement learning level. PI is the average of the accumulated rewards (score) of the last 10 episodes. Each worker sends the calculated gradients and PI to the chief. The chief calculates the weighted arithmetic mean to take into account the PI of each worker, unlike the previous study, which simply averaged the gradients. The averaged gradients of each worker are not simply exchanged, but are weighted according to each worker's degree of learning. The weights are calculated as follows (Figure 2a's ①):

$$w^n = \frac{PI_\theta^a}{\sum_{n=1}^{N} PI_\theta^n} \tag{8}$$

where PI_θ^a is an agent's Performance Index, w^n is an individual worker's weight, N is the number of workers and PI_θ^n is an individual worker's PI. The chief calculates the weights for all workers, and the weighted arithmetic mean is calculated using them. The weighted arithmetic mean is obtained as follows (Figure 2a's ②):

$$\bar{g} = \sum_{n=1}^{N} w^n g_\theta^n \tag{9}$$

where $\bar{g}$ is the average gradient obtained by the weighted arithmetic mean and g_θ^n is each worker's gradient; w^n is [0, 1] in an agent, and all agents' weighted sum is $\sum_{n=1}^{N} w^n = 1$ Finally, the chief sends the $\bar{g}$ to all workers. Each worker updates the π_θ with $\bar{g}$.

After performing several episodes, the worker satisfies the termination condition and completes learning. At that same time, the next episode is performed for the model transfer learning. The parameters of the mature Actor model that completed the learning process are sent to the chief (Figure 2b's ①). The chief sends the mature actor model parameters to the rest of the workers (Figure 2b's ②). The other workers receive mature parameters and replace their own Actor model parameters. However, instead of entirely replacing them with the mature parameters received from the chief, the transfer weight is calculated and appropriately replaced according to the PI indicating the learning performance for each worker. The transfer weight indicates to which extent the PI meets the termination conditions for each environment. The transfer weights w^T are calculated as follows:

$$w^T = \frac{PI_t}{TC} \tag{10}$$

where PI_t is the Performance Index at time step t and TC represents the condition to end learning in each environment. The worker considers its own Actor model parameters θ and replaces them with mature parameters received from the chief $\bar{\theta}$ according to the calculated w^T. The replacement formula is as follows:

$$\theta = w^T \times \theta + (1 - w^T) \times \bar{\theta} \tag{11}$$

With Actor–Critic PPO, the proposed federated reinforcement learning algorithm for acceleration is provided in Algorithms 1 and 2. The parameter M that represents the maximum number of episodes is shared by all workers and the

Algorithm 2 Federated RL (Worker n)

```
for $i = 1, 2, 3, \ldots, M$ do
    for each step $t$ of an episode do
        Start the actor model $\pi_\theta$ 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 for
    if learning process is finished then
        Send to the chief a message $m_n$ including the
        actor model parameter $\theta_n$
        Break
    else
        Update $\pi_{\theta_{old}} \leftarrow \pi_\theta$
        for $j = 1, 2, 3, \ldots, K$ do
            Get a mini-batch $B$ from the trajectory
            memory (the size of $B$ is $U$)
            for $t = 1, 2, \ldots, U$ do
                Compute $\widehat{A}_t$, $L_t^V(\mu)$ and $L_t^{CLIP}(\theta)$ by
                using Equation (2),(3),(6)
            end for
            Compute the gradient $g_\mu = \nabla L^V(\mu)$ for the
            critic model parameter $\mu$
            Update $V_\mu$ with $g_\mu$ through SGD
            Compute the gradient $g_\theta = \nabla L^{CLIP}(\theta)$ for
            the actor model parameter $\pi$
            Update $\pi_\theta$ with $g_\theta$ through SGD
            Append the cumulative reward to $PI$
        end for
        Compute average the latest 10 episode's $PI$
        Send to the chief a message $m_n$ including the last
        gradient $g_\theta^n$ and $PI$
    end if
    if 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
        Compute the $w^T$ by using Equation (10)
        $\theta = w^T \times \theta + (1 - w^T) \times \bar{\theta}$
    else if $m_c$ has the average gradient $\bar{g}$ then
        Update $\pi_\theta$ with the received $\bar{g}$ through SGD
end for
```

chief. The chief retains the set of all workers N. Whenever the chief receives the Actor model parameters θ_n from a worker n, the chief removes it from N when the episode is completed. In a worker, K is the number of optimizations in one episode.

V. EXPERIMENTS

In this section, we verify the efficacy of the proposed federated reinforcement learning by applying it to a simulation environment and a real device. To verify the efficacy of the proposed federated reinforcement learning, we compare weighted-based gradient sharing and transfer learning with unweighted ones. The simulation environment uses OpenAI Gym, and the real device uses Quanser$^{\text{TM}}$'s QUBE-Servo 2.

A. EXPERIMENTS CONFIGURATION

The experimental system's configuration for controlling the simulation environments and the real device contains four workers and one chief. The workers and chief are installed on Ubuntu 18.04 LTS version and for our Actor-Critic based on PPO algorithm we used the Python 3.6 and PyTorch 1.2 version. The Actor and Critic models consist of three multi-layer perceptron where each layer includes 128 neurons and two separate output layers respectively. The output layer of Actor models takes output size (i.e., action space according to environments), the output layer of Critic model takes single value to evaluate the chosen action by Actor model. Also, we use the hyper-parameters of the Actor-Critic PPO which the clipping parameter of which is 0.9, and GAE parameter is 0.99. The model optimization is Adam optimizer, Actor and Critic model's learning rates are 0.001, trajectory memory size is 400, and batch size is 128. If the trajectory memory is small, the reinforcement learning model is updated using the model's experience (state, behavior, reward, next state) collected at the latest steps. However, if the trajectory memory is large, the previous experiences are also considered and updated. In general, the PPO model tends to be reliably updated which is performed based on the experience collected through the recently updated model. The maximum number of episodes M is 2000, and the number of workers N is 4 in the simulation environment, and 3 in the real device environment.

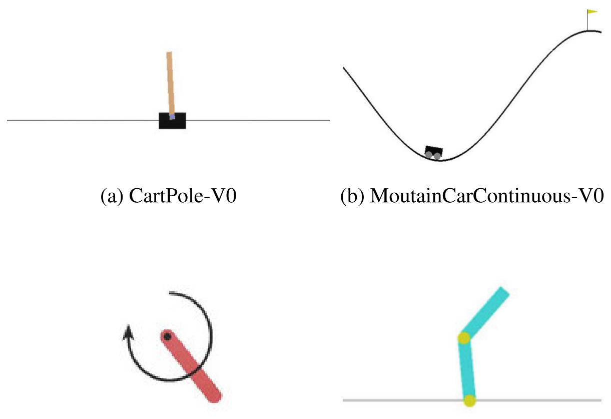
B. EXPERIMENTS ENVIRONMENTS

1) SIMULATION ENVIRONMENTS

In order to verify the proposed federated reinforcement learning, the simulation environments are selected from OpenAI Gym. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a variety of simulation environments and supports easy modification and uses for testing reinforcement learning. For experiment, we adopted CartPole, MountainCarContinuous, Pendulum, and Acrobot. It corresponds to the classic control problem which is commonly used in the field of control. Figure 3 shows four simulation environments to evaluate the performance of proposed federated reinforcement learning.

The target of CartPole's is to keep the pole upright on the cart and the cart stops the pole from toppling by moving from side to side with controlling speed on the frictionless track. CartPole has four observations: cart position (Min -2.4 to Max 2.4), cart velocity, pole angle (Min $-41.8°$ to Max$41.8°$), and pole velocity at the tip. It has two discrete actions: push the cart left or right and the reward is $+1$ for every step taken, including the termination step. The pole angle is more than $\pm 12°$, which is the episode's termination condition. The cart is at or beyond ± 2.4, or the total sum of rewards in one episode is 200. If the average reward for episodes of 10 consecutive learning attempts is 195.0 or higher, the conditions to conclude the learning episode are considered to have been met.

MountainCarContinuous's subject is an underpowered car that must summit the one-dimensional hill on the right to



(a) CartPole-V0 (b) MoutainCarContinuous-V0

(c) Pendulum-V0 (d) Acrobot-V1

FIGURE 3. OpenAI Gym simulation environments for experiments.

reach a target. There is another hill on the left and as the car ascends it, potential energy is developed and on release, the car accelerates towards the flag. MountainCarContinuous has two observations: car position (Min -1.2 to Max 0.6), car velocity (Min -0.07 to Max 0.07). MountainCarContinuous has continuous actions: push the car to the left (negative value) or the right (positive value). The reward is 100 when the flag on the hill is reached, minus the sum of the squared actions from the beginning of the episode, until the goal is attained. The episode's termination condition is that the car's position equals 0.5. The end condition of the learning phase is reached when a reward of more than 200 is obtained in 10 consecutive learning attempts.

Acrobot has two joints and two links and the joints between the links are actuated. Initially, the links hang down and the Acrobot's objective is to swing the end of the lower link up to a given height. Acrobot has six observations: the rotational joint angles and velocities of joints and links. Acrobot has discrete actions: torque on the joint between the two pendulum links, effort to the left or right, and stop. The reward is -1 for each step in each episode and 0 when the second link reaches the target height. The episode's termination condition is that the second link reaches the target height. The end condition is to obtain an average of more than -100 in 10 consecutive learning attempts.

Pendulum's objective is to keep a frictionless pendulum vertical. Pendulum upwardly rotates the pendulum that is tilted downward with force from the right or left. Pendulum has three observations: pendulum angle (cos and sin values) and pendulum angular velocity. Pendulum has continuous action: the joint effort (between -2.0 and 2.0) to the left (negative value) or the right (positive value). The reward is the following:

$$r_t = -(\theta^2 + 0.1 \times \bar{\theta}^2 + 0.001 \times action^2) \tag{12}$$

where θ is the normalized pendulum angle between -π and π, and $\bar{\theta}$ is the pendulum's angular velocity. Therefore, the lowest cost is $-(\pi^2 + 0.1 \times 8^2 + 0.001 \times 2^2) = -16.2736044$, and the highest cost is 0. The episode's termination condition is when 200 steps are attained for each episode. The end condition is that the rewards in 10 consecutive learning attempts average more than -400.

Additionally, we added noise to the four simulation environments for the experiment to mimic a real device environment. Because, even if real devices are produced in the same factory, their physical and dynamic characteristics may be different. In other words, even if the same torque is given, and the motor is activated to run, the actual torque of the motor may be different for each machine. Therefore, we measured the noise for each of the QUBE-Servo 2 devices that were used. We set the motor's torque to ± 15 and measured the change in the motor angle 100 times. Although the same torque is given to the real device, there is an average difference of $\pm 0.002°$, and a standard deviation of $\pm 0.0005°$ per machine. Therefore, we experimented by adding noise to each action in the simulation environment according to the Gaussian distribution with an average of 0.002 and a standard deviation of 0.0005.

2) REAL-DEVICE ENVIRONMENTS

We use QUBE-Servo 2s as the RIP devices for the real device environment. The RIP system is used in mechanical control to present classic control system problems. Our QUBE-Servo 2 is an unstable nonlinear RIP device that has commonly been used in the field of engineering nonlinear mechanical controls. The objective of the QUBE-Servo 2 is to balance a rigid pendulum vertically The QUBE-Servo 2 has four types of observation: pendulum angle, pendulum angular velocity, motor angle, and motor angular velocity. The action is chosen from the actor model of each worker's actor-critic PPO. Also, in order to keep the pendulum vertical, the action must be selected and applied within 7ms. The chosen action is -60, 0, or 60, which signifies turn left, stop, and turn right. The reward is $+1$ for every step in one episode. This is because the RL agent maintained balance during the step. However, if the pendulum's angle is outside the range of $\pm 7.5°$, the reward is 0. The termination condition for the episode is when the pendulum angle fell out of the $\pm 7.5°$ range and balance failed When the rewards in one episode reached 2450 or more, the episode ended. The QUBE-Servo 2 must perform an action within 7ms to keep the pendulum vertical. So, getting 2450 rewards for one episode is keeping the balance of pendulum for about 15 seconds. The termination condition for learning is that the average of the rewards in the last 10 episodes is 2450 or more.

For our proposed federated reinforcement learning, the experimental configuration in the real environment consisted of three workers and one chief as shown in Figure 4. QUBE-Servo 2 SPI-port cannot connect directly to the switch, hence we need a Raspberry Pi for only physical connection between the switch and QUBE-Servo 2. QUBE-Servo 2 interacted with the worker's actor-critic PPO agent via Raspberry Pi using Serial Peripheral Interface (SPI)

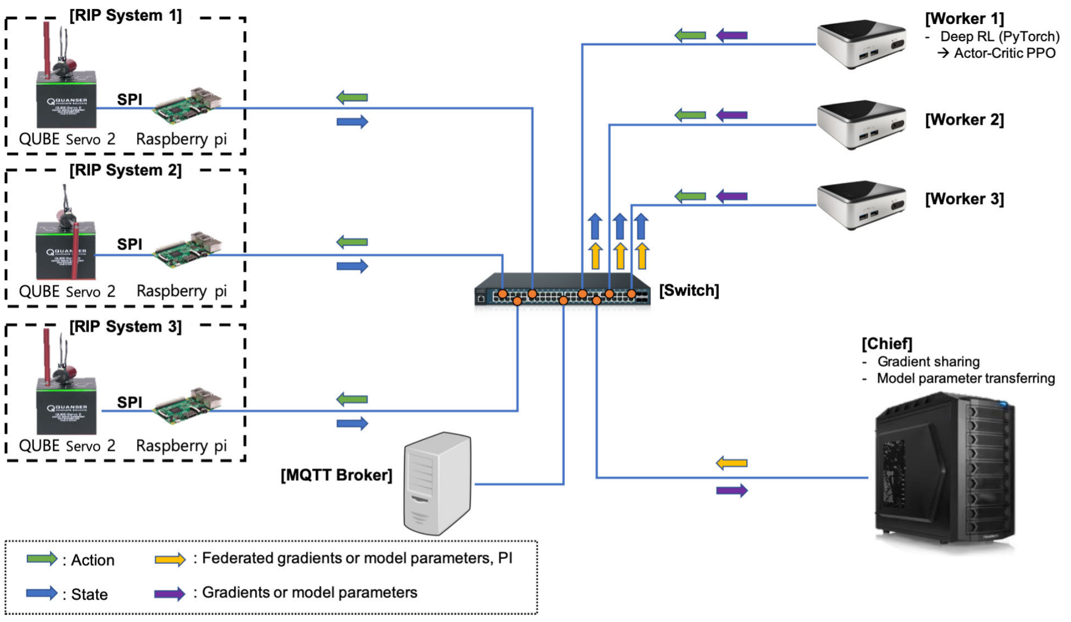
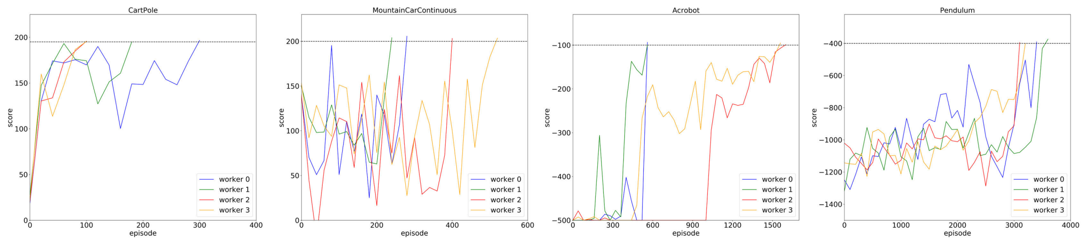
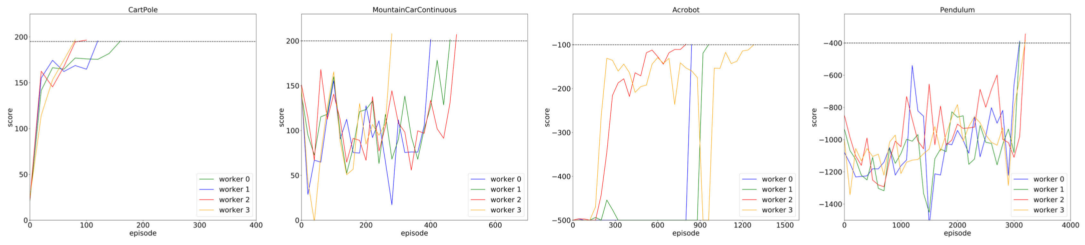


FIGURE 4. The system architecture of federated reinforcement learning experiment.



(a) Ordinary federation policy



(b) Proposed federation policy

FIGURE 5. Effectiveness of the proposed federated reinforcement learning methods in simulation environments. The blue, green, red and yellow line are the lastest 10 average accumulated rewards of each worker, and the black dotted line is the reward of termination condition for each simuation environment.

communication. The Raspberry Pi receives state information (i.e., pendulum angle, pendulum angular velocity, motor angle, and motor angular velocity) from the QUBE-Servo 2, and forwards them to the actor-critic PPO agent in a worker. Also, it receives the chosen action (i.e., motor power) from the actor-critic PPO agent in a worker, converts it into a voltage value, and eventually forwards it to the QUBE-Servo 2. And we use the MQTT protocol for communication between the RIP system, workers, and chief. The MQTT protocol requires a broker in the middle and interacts between the RIP system and workers through the MQTT broker. The worker and the RIP system exchange state information and actions with each other, so that the worker performs training. In addition, we use MQTT to exchange gradients, accumulated rewards (PI), and model parameters between the chief and workers.

C. EFFECT PROPOSED FEDERATED REINFORCEMENT LEARNING IN SIMULATION ENVIRONMENTS

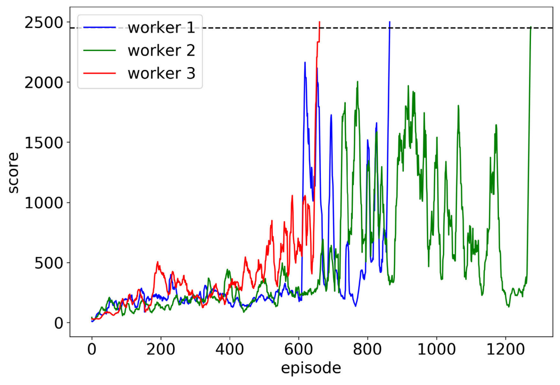
Figure 5 shows the effectiveness of the proposed federated reinforcement learning methods in simulation environments. In particular, the federation policies proposed in this paper represent the effect of weight-based gradient sharing and weight-based transfer learning. In the experiment, we used the system architecture in Figure 1 and performed federated reinforcement learning using four workers and one chief. Figure 5a is the result of simply performing gradient sharing and transfer learning in four simulations without being weight-based. In contrast, Figure 5b is the result of weight-based gradient sharing and transfer learning using the proposed federation policy. The experiment is performed 10 times for each simulation environment, and the graphs in Figures 5a and 5b represent the average accumulated rewards of the last 10 episodes divided by each worker.

As shown in Figure 5, the learning speed of federated reinforcement learning using weight-based gradient sharing and transfer learning, which is the proposed federation policy, is generally high. In the case of CartPole, using the proposed federation policy, 160 is the workers' last episode. When used without the proposed federation policy, the final worker's last episode is 310. For the federated reinforcement learning that applied the federation policy proposed for MountainCarContinuous, Acrobot, and Pendulum, the learning respectively ended at episodes 455, 1222, and 3196. Conversely, when used without the proposed federation policy, the learning ended at episodes 517, 1654, and 3592.

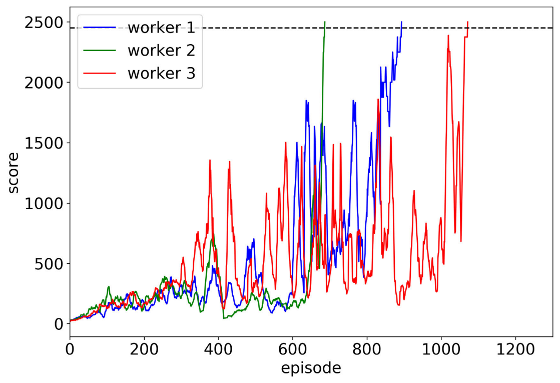
The reason for the high performance of federated reinforcement learning by applying the proposed federation policy is that different noises were added to the simulation environment as happens in the real world. In other words, in the case of exchanging the gradients or parameters of a completed model that is an entire learning experience, less learning time is required due to a subtle difference in the environment. Therefore, in the case of the proposed federation policy, the weight is assigned using PI, which represented the level of the current learning experience, therefore, if the learning is well executed, the current learning experience could be maintained. Also, in the case of a worker that did not learn effectively, the weight is low, therefore, it reflects many new experiences and learns well. This trend is evident in Acrobot. In addition, the number of episodes required to terminate the remaining workers at the time of transfer from the completed learning model is generally low in the proposed federated reinforcement learning system.

D. EFFECT OF PROPOSED FEDERATED REINFORCEMENT LEARNING IN REAL ENVIRONMENTS

Figure 6 shows that the effect is verified by applying the proposed federated reinforcement learning to several QUBE-Servo 2s, which represent real environments. Unlike the simulation environment, this experiment uses 3 workers, that is, three QUBE-Servo 2s. The rest of the experiment configuration is the same as the simulation environment.



(a) Changes of the lastest 10 average accumulated rewards without proposed federation policy



(b) Changes of the lastest 10 average accumulated rewards with proposed federation policy

FIGURE 6. Effectiveness of the proposed federated reinforcement learning methods in real envrionments as RIP system. The blue, green, and red line are the lastest 10 average accumulated rewards each worker, and the black dotted line is the reward for the termination condition (2450) for each RIP system.

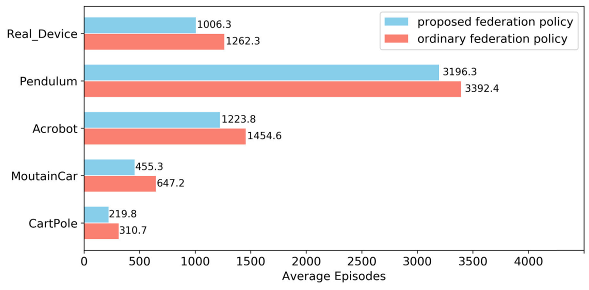


FIGURE 7. Average of episodes where learning is performed 10 times for the simulation and real environment respectively.

The termination condition in the real environment is 2450 and is indicated by a black dotted line. Figure 6a is the result of federated reinforcement learning without the proposed federation policy, and Figure 6b is the opposite. When the proposed Federation policy is applied, all workers are completed learning in 1071 episodes. In the other case, learning is completed after 1273 episodes. Similarly, to the experimental results from the simulation environment, it can be seen that a low number of episodes is required for a worker's learning to be completed and the rest of the workers' learning to be completed.

Figure 7 shows the average number of episodes in which learning ended after respectively applying and not applying the proposed federation policy 10 times in the simulation and real environment. For all experimental environments, learning speed is high when federated reinforcement learning with the proposed federation policy is applied.

VI. CONCLUSION

In this paper, we have shown that the proposed federated reinforcement learning can successfully control multiple simulation environments and real devices with slightly different dynamics. We used Actor-Critic PPO that demonstrates good performance as a reinforcement learning algorithm and applied a new federation policy. The proposed federation policy is weight-based, gradient sharing, and transfer learning that more rapidly solved the classical control problem environments of OpenAI Gym. Although, physical noise may exist in real devices, the proposed federation policy can reliably learn multiple devices at the same time, and achieve the optimal goal with fewer training times. Our approach improves learning performance by approximately 1.2 times compared to a previous real-environment study. In future work, we will apply this approach to a more complex real device such as a double RIP system. Moreover, we plan to research new reinforcement learning techniques to achieve optimal performance with less training. The proposed federation policy will be applied to various algorithms such as DQN and DDPG, and comparative verification will be performed with multi-agent based reinforcement learning such as QMIX and QTRAN. We also have planed to study new reinforcement learning techniques to achieve optimal performance with less training. We will extend our research to reduce the communication delay between the agents and the environment. However, there are several limitations if this research applied to a real network, such as Software-Defined Networking (SDN) and Virtual Network Embedding (VNE). Hence, more research and efficient distributed-based multi-agent RL are needed for smooth communication and better performance.

REFERENCES

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

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[3] 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, 2015.

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. ICML*, 2016.

[5] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement learning and deep learning based lateral control for autonomous driving," 2018, *arXiv:1810.12778*. [Online]. Available: http://arxiv.org/abs/1810.12778

[6] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, pp. 1–15, Apr. 2017.

[7] Y. Xu, Z. Deng, M. Wang, W. Xu, A. M.-C. So, and S. Cui, "Voting-based multiagent reinforcement learning for intelligent IoT," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2681–2693, Feb. 2021.

[8] K. Kersandt, G. Munoz, and C. Barrado, "Self-training by reinforcement learning for full-autonomous drones of the future*," in *Proc. IEEE/AIAA 37th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2018, pp. 1–10.

[9] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous Internet of Things: Model, applications and challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1722–1760, 3rd Quart., 2020.

[10] J. Konecný, H. B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," 2015, *arXiv1511.03575*. [Online]. Available: https://arxiv.org/abs/1511.03575

[11] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," 2019, *arXiv:1910.06001*. [Online]. Available: http://arxiv.org/abs/1910.06001

[12] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4555–4562, Oct. 2019.

[13] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[14] S. Kumar, P. Shah, D. Hakkani-Tur, and L. Heck, "Federated control with hierarchical multi-agent deep reinforcement learning," 2017, *arXiv:1712.08266*. [Online]. Available: http://arxiv.org/abs/1712.08266

[15] J.-B. Kim, H.-K. Lim, C.-M. Kim, M.-S. Kim, Y.-G. Hong, and Y.-H. Han, "Imitation reinforcement learning-based remote rotary inverted pendulum control in OpenFlow network," *IEEE Access*, vol. 7, pp. 36682–36690, 2019.

[16] H.-K. Lim, J.-B. Kim, J.-S. Heo, and Y.-H. Han, "Federated reinforcement learning for training control policies on multiple IoT devices," *Sensors*, vol. 20, no. 5, p. 1359, Mar. 2020.

[17] H.-K. Lim, J.-B. Kim, S. Y. Kim, and Y.-H. Han, "Federated reinforcement learning for automatic control in sdn-based iot environments," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 1868–1873.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[19] V. Konda, "Actor-critic algorithms," Ph.D. dissertation, Cambridge, MA, USA, 2002.

[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2017, pp. 6382–6393.

[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.* in Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, Jul. 2015, pp. 1889–1897.

[22] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2019, *arXiv:1911.10635*. [Online]. Available: http://arxiv.org/abs/1911.10635

[23] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, vol. 54, 2017, pp. 1273–1282.

[24] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. M. Kiddon, J. Konecný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: https://arxiv.org/abs/1902.01046

[25] H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated reinforcement learning," 2019, *arXiv:1901.08277*. [Online]. Available: https://arxiv.org/abs/1901.08277

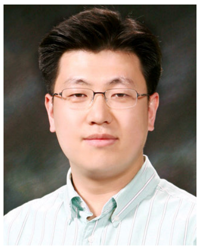
[26] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 7, pp. 1633–1685, 2009.
[27] F. L. Da Silva and A. H. R. Costa, "Transfer learning for multiagent reinforcement learning systems," in *Proc. 25th Int. Joint Conf. Artif. Intell.* Menlo Park, CA, USA: AAAI Press, 2016, pp. 3982–3983.
[28] R. Glatt, F. L. Da Silva, and A. H. R. Costa, "Towards knowledge transfer in deep reinforcement learning," in *Proc. 5th Brazilian Conf. Intell. Syst. (BRACIS)*, Oct. 2016, pp. 91–96, doi: 10.1109/BRACIS.2016.027.
[29] P. Vrancx, Y.-M. De Hauwere, and A. Nowé, "Transfer learning for multi-agent coordination," in *Proc. 3th Int. Conf. Agents Artif. Intell.*, Rome, Italy, 2011, pp. 263–272.
[30] F. L. D. Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *J. Artif. Intell. Res.*, vol. 64, pp. 645–703, Mar. 2019.
[31] D. Q. Tran and S.-H. Bae, "Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection," *Appl. Sci.*, vol. 10, no. 16, p. 5722, Aug. 2020.
[32] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*. [Online]. Available: https://arxiv.org/abs/1506.02438
[33] D. V. Lindley, "Information theory and statistics. Solomon Kullback," *J. Amer. Stat. Assoc.*, vol. 54, no. 288, pp. 825–827, 1959.

HYUN-KYO LIM received the B.S. degree in computer science and engineering and the M.S. degree in computer science engineering from the Korea University of Technology and Education, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the Department of Interdisciplinary Program in Creative Engineering. He studied mobility management during his master course and he especially researched distributed mobility management in software-defined networking. He is studying deep learning and reinforcement learning during his doctoral studies. He is also exploring ways to apply deep learning and reinforcement learning to the network and is working on applying deep learning and reinforcement learning to a variety of applications.

JU-BONG KIM received the B.S. and M.S. degrees in computer science and engineering from the Korea University of Technology and Education, Cheonan, South Korea, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering. His research interests include deep learning and reinforcement learning across various domains, especially device control and blockchain assets.

IHSAN ULLAH received the B.S. and M.S. degrees in computer science from the University of Peshawar, Pakistan, in 2001 and 2004, respectively, and the Ph.D. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019. From September 2019 to August 2020, he was a Postdoctoral Research Fellow with the Ubiquitous Computing Technology Research Institute (UTRI), Sungkyunkwan University. Since 2020, he has been a Research Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. His research interests include data aggregation, data fusion, virtual network embedding, network slicing (5G), the Internet of Things (IoT), artificial intelligence, machine learning, cloud computing, and wireless sensor networks.

JOO-SEONG HEO received the B.S. and M.S. degrees in computer science and engineering from the Korea University of Technology and Education, Cheonan, South Korea, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree with the Interdisciplinary Program in Creative Engineering. His research interests include deep learning and reinforcement learning, device control, and blockchain assets.



YOUN-HEE HAN (Member, IEEE) received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in computer science and engineering from Korea University, Seoul, South Korea, in 1996, 1998, and 2002, respectively.

From 2002 to 2006, he was a Senior Researcher with the Next Generation Network Group, Samsung Advanced Institute of Technology. Since 2006, he has been a Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. Since 2002, his activities have been focusing on mobility management, media independent handover, and cross-layer optimization for efficient mobility support. He has published approximately 250 research articles on the theory and application of mobile computing and has filed 40 patents on information and communication technology domain. His current research interests include theory and application of computer networks, including protocol design and mathematical analysis, mobile sensor/actuator networks, social network analysis, machine learning, deep learning, and reinforcement learning. He has made several contributions in IETF and IEEE standardization. He has served as the Co-Chair for working group in the Korea TTA IPv6 Project Group. He has been serving as an Editor for the *Journal of Information Processing Systems* since 2011.