

Article

Exploration with Multiple Random ϵ -Buffers in Off-Policy Deep Reinforcement Learning

Chayoung Kim ¹ and JiSu Park ^{2,*}¹ Division of General Studies, Kyonggi University, Suwon, Gyeonggi-do 16227, Korea; kimcha0@kgu.ac.kr² Convergence Institute, Dongguk University, Seoul 04620, Korea

* Correspondence: bluejisu@dgu.edu; Tel.: +82-2-2290-1491

Received: 26 September 2019; Accepted: 30 October 2019; Published: 1 November 2019



Abstract: In terms of deep reinforcement learning (RL), exploration is highly significant in achieving better generalization. In benchmark studies, ϵ -greedy random actions have been used to encourage exploration and prevent over-fitting, thereby improving generalization. Deep RL with random ϵ -greedy policies, such as deep Q-networks (DQNs), can demonstrate efficient exploration behavior. A random ϵ -greedy policy exploits additional replay buffers in an environment of sparse and binary rewards, such as in the real-time online detection of network securities by verifying whether the network is “normal or anomalous.” Prior studies have illustrated that a prioritized replay memory attributed to a complex temporal difference error provides superior theoretical results. However, another implementation illustrated that in certain environments, the prioritized replay memory is not superior to the randomly-selected buffers of random ϵ -greedy policy. Moreover, a key challenge of hindsight experience replay inspires our objective by using additional buffers corresponding to each different goal. Therefore, we attempt to exploit multiple random ϵ -greedy buffers to enhance explorations for a more near-perfect generalization with one original goal in off-policy RL. We demonstrate the benefit of off-policy learning from our method through an experimental comparison of DQN and a deep deterministic policy gradient in terms of discrete action, as well as continuous control for complete symmetric environments.

Keywords: deep Q-network (DQN); reinforcement learning (RL); explorations; deep deterministic policy gradient (DDPG); random ϵ -greedy buffers

1. Introduction

Deep reinforcement learning (RL) [1] has been applied in challenging domains, such as games and robotics. A combination of RL and nonlinear function approximates, such as deep neural networks, helps in automating decision making and control issues. However, it can also be challenging in terms of stability and convergence [2] in real-time online learning, such as in the detection of network security through the verification of whether the network is “normal or anomalous” [3–7]. Moreover, its widespread adaptation in the real world, such as with robotic arms, is difficult because of sample complexity and heavy dependence on certain hyper-parameters, such as exploration constants [3–7]. A significant challenge in robotics is the engineering of a carefully shaped reward function. However, the necessity of reward function engineering restricts the general applicability of RL in the real world. Recent remarkable research on hindsight experience replay (HER) [8] provides additional replay buffers with the original goal and a subset of other goals without any domain knowledge. HER improves the sample-efficient setting and learns the possible multi-goals, even in an environment of sparse rewards. Off-policy algorithms in RL [1], such as deep Q-networks (DQNs) [9,10], reuse the past experience with random ϵ -greedy actions in replay buffers for efficient exploration behaviors. Andrychowicz et al. [8] found that a DQN combined with HER resolves some tasks more easily than that without

HER. An experience comprises a tuple, such as state, action, reward, or new state, and each agent of DQN can exploit a batch of experiences to update the Q-value function via temporal differencing (TD) learning [9,10]. The experience replay (ER) buffer discards old memories at each step by sampling the buffer randomly and updating the DQN agent. It aids in breaking temporal relations and increasing data usage [11,12].

The present study is inspired by the quantification of generalization in RL [11], which investigates the impact of injecting stochastic methods on generalization by random ϵ -greedy actions overriding the agent's preferred actions. A random ϵ -greedy action has been used to encourage exploration and prevent excessive over-fitting [11]. There has been research on the exploration diversities of stability and convergence. A prioritized replay memory [13] has been used in a DQN for replaying important transitions more frequently and learning more efficiently. HER [8] makes possible sample-efficient learning with additional replay buffers. A parameter space in noise [14] in a neural network leads to a more abundant set of exploratory behaviors through noise injection in the action space in deep RL. However, some empirical implementations show that in certain environments, adding parameter space noise [14] may not improve ϵ -greedy random actions by OpenReivew.net [15]. A prioritized replay memory attributed to a complex TD error might provide better theoretical results. The TD errors might create a balance issue between variance and bias, because TD attributes can be computed at every n -time step [11,12]. Moreover, some studies have only partially succeeded in combining drop-out, L2 regularization, and ϵ -greedy random actions. Thus, training with only ϵ -greedy random actions can vastly improve generalization and exploration [11,12].

This paper proposes multiple ϵ -greedy experience buffers in off-policy deep RL [1] for enhancing explorations for a more near-perfect generalization for the original goal. We consider strengthening the advantages of exploring model-free, deep RL, and demonstrate that the off-policy method benefits through the experimental comparison of DQN [9,10] and DDPG [16], which is actually based on on-policy, and represents a trade-off between policy optimization [2,16] and Q-learning [9,10]. The proposed deep RL model is compatible with the discrete action as well as continuous control, symmetrically. Therefore, we can expect better prediction accuracy in real-time online learning, such as detecting network intrusions by verifying whether the network is "normal or anomalous" [3–7]. The concept of our proposed multiple ϵ -buffers is extremely simple. After experiencing some episodes, we store them in the replay buffers R_1 and R_2 . Note that we can replay one trajectory with only one goal, assuming that we exploit an off-policy RL like DQN [9,10] and DDPG [16]. When the procedure passes through multiple ϵ -buffers, the trade-off between policy optimization and Q-learning can be solid and strong due to the deep neural network in DQN [9,10] or DDPG [16].

2. Background

RL comprises an agent with an environment that is described by a set of states, S , a set of actions, A , a distribution of initial states, $p(s_0)$, a reward function, $r: S \times A \rightarrow R$, transition probabilities, $p(s_{t+1}|s_t, a_t)$, and a discount factor, $\gamma \in [0,1]$ [1]. Deterministic policy maps from states to actions are given as $\pi: S \rightarrow A$. At each time step, t , the agent selects an action attributed to the current state: $a_t = \pi(s_t)$, and takes the reward, $r_t = r(s_t, a_t)$ [1]. The new state of the environment is sampled from the distribution $p(|s_t, a_t)$. A discounted cumulative reward is a return, $R_t = \sum T \gamma^{t'-t} r_{t'}$, where T is the time step at which the agent's simulation terminates [1]. The goal here is to maximize its expected return $E_{s_0}[R_0|s_0]$. The action value function, named Q-function, is defined as $Q^\pi(s_t, a_t) = E[R_t|s_t, a_t]$. An optimal policy is denoted by the following equation, called the Bellman equation, $Q^*(s, a) = E_{s' \sim p(|s, a)} [r + \gamma \max_{a'} Q^*(s', a')|s, a]$, (1), this equation converges at the Q-function as an optimal action value [1]. The Bellman equation is to be used as a function approximation for estimating the Q-function, $Q(s, a; \theta) \approx Q^*(s, a)$, as an action value [1]. This can be a linear-function approximation, but it is sometimes even a nonlinear-function approximation, such as with a deep neural network [9,10]. Therefore, we attempt to exploit a deep neural network as a function approximation. We refer to a deep neural network approximation with weights θ as the Q-function [9,10]. A Q-function network can be

trained by minimizing a sequence of loss functions, $L_i(\theta_i)$, which changes at i per iteration [1,9,10]. $L_i(\theta_i) = E_{s,a \sim p(s,a)} [(y_i - Q(s,a;\theta_i))^2]$, (2), where $y_i = E_{s' \sim p(s,a)} [r + \gamma \max_{a'} Q^*(s',a';\theta_{i-1}) | s,a]$ is the target for i per iteration. The loss function can be differentiated with regard to the weights, depending on the gradient [1,9,10]. $\nabla \theta_i L_i(\theta_i) = E_{s,a \sim p(s,a); s' \sim \epsilon} [r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)] \nabla \theta_i Q(s,a;\theta_i)$, (3), [1,9,10]. Optimizing the loss function by using the stochastic gradient descent is computationally intricate. Q-learning as a function approximation becomes possible [17,18], and is model-free, which indicates the RL model and off-policy. This implies that it learns about greedy strategy $a = \max_a Q(s,a;\theta)$ while obeying an exploration selected by ϵ -greedy with a probability of $1 - \epsilon$ and a random action with a probability of ϵ for exploration and exploitation [17,18]. DQN is a well-known, model-free RL algorithm attributed to discrete action spaces. In DQN [9,10], we construct a deep neural network, Q , which approximates Q^* and is greedy-defined as $\pi_Q(s) = \arg \max_{a \in A} Q(s,a)$ [9,10]. It is a ϵ -greedy policy with probability ϵ and takes the action $\pi_Q(s)$ with probability $1 - \epsilon$. Each episode uses the ϵ -greedy policy following Q as an approximation of the deep neural network. The tuples (s_t, a_t, r_t, s_{t+1}) are stored in the replay buffer, and each new episode is configured to neural network training [9,10]. The deep neural network is trained using the gradient descent of random episodes on loss L , encouraging Q to follow the Bellman equation [9,10]. The tuples are sampled from the replay buffer of random episodes. The target network y_t is computed by a separate neural network that changes more slowly than the main deep neural network in order to optimize a stable process. The weights of the target network are set to the current weights of the main deep neural network. The DQN algorithm [9,10] is presented in Algorithm 1.

Algorithm 1 Deep Q-learning with Experience Replay

```

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M do
    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
    for t = 1, T do
        With probability  $\epsilon$ , select a random action  $a_t$ 
        Otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in D
        Sample random mini-batch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from D
        Set  $y_j = r_j$  for terminal  $\phi_{j+1}$ 
        Or  $y_j = r_j + \gamma \max_{a'} Q^*(\phi_{j+1}, a'; \theta)$  for non-terminal  $\phi_{j+1}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j | \theta))^2$ 
    end for
end for
  
```

A deep deterministic policy gradient (DDPG) [16] is a well-known, model-free RL algorithm attributed to continuous action spaces [2,16]. In a DDPG, we construct two neural networks: a target policy as an actor, $\pi: S \rightarrow A$, and an action-value-function approximation as a critic, $Q: S \times A \rightarrow R$. The critic approximates Q^π , which is the action-value function of the actor [16]. Each episode uses a noisy policy of the target, $\pi_b(s) = \pi(s) + N(0,1)$ [16]. The critic is trained in the same way as the Q-function in a DQN [9,10]. The target network y_t is computed using the output of the actor, i.e., $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$, and the actor's deep neural network is trained using the gradient descent of random episodes on the loss $L_a = -E_s Q(s, \pi(s))$ [2,16] and sampled from the replay buffer of random episodes. The gradient of L_a is computed by both the critic and actor through back-propagation [2,16]. The algorithm of a DDPG [16] is shown in Algorithm 2.

Algorithm 2 Deep Deterministic Policy Gradient

Randomly initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay memory R
for episode = 1, M do
 Initialize a random process N for action exploration
 Receive initial observation state s_1
 for $t = 1, T$ do
 Select action $\alpha_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise
 Execute action α_t and observe reward r_t and new state s_{t+1}
 Store transition $(s_t, \alpha_t, r_t, s_{t+1})$ in R
 Sample **random mini-batch of transitions** $(s_j, \alpha_j, r_j, s_{j+1})$ from R
 Set $y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'}))|\theta^{Q'}$
 Update critic by minimizing the loss: $L = \frac{1}{n} \sum_j (y_j - Q(s_j, \alpha_j|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:
 $\nabla_{\theta^\mu} L \approx \frac{1}{n} \sum_j \nabla_{\alpha} Q(s, \alpha|\theta^Q)|_{s=s_j, \alpha=\alpha_j} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_j}$
 Update the target:
 $\theta^{Q'} \leftarrow \tau \theta^{Q'} + (1-\tau) \theta^Q$
 $\theta^{\mu'} \leftarrow \tau \theta^{\mu'} + (1-\tau) \theta^\mu$
 end for
end for

3. Multiple Random ϵ -Buffers**3.1. Proposed Off-Policy Algorithm**

We propose multiple random ϵ -greedy buffers in a DQN [9,10] for discrete action spaces and DDPG [2,16] for continuous action spaces. The idea of our proposed multiple ϵ -buffers is to store experienced episodes in the replay buffers R_1 and R_2 . For better exploration, the agent evaluates a target policy $\pi(a|s)$ to compute $V\pi(s)$ or $Q\pi(s,a)$ while following a behavior policy $\mu(a|s)$, where $\{s_1, a_1, r_2, \dots, s_T\} \sim \mu$. The agent learns about multiple policies while following one policy. This means that it can learn about the optimal policy and follow the exploratory policy. The target policy π is greedy with respect to $Q(s,a)$, such as $\pi(s_{t+1}) = \operatorname{argmax}_{a'} Q(s_{t+1}, a')$, while the behavior policy μ is ϵ -greedy with respect to $Q(s,a)$, such as $\mu(s_{t+1}) = R_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a')$. When an agent follows a greedy policy, convergence changes rapidly. However, because of a lack of exploration, the agent reaches local optimization. Many other challenges address this global-local dilemma [2,9,10,16]. A remarkable recent work, HER [8], provides additional replay buffers with the original goal and a subset of other goals without any complicated reward function. The off-policy algorithms in RL, such as DQN [9,10] or DDPG [2,16], reuse the past experience with random ϵ -greedy actions in replay buffers for better exploration. The ER buffers discard old memories at each step by sampling the buffer randomly to update the DQN agent. Therefore, ER helps to break temporal relations and increase data usage [11,12].

Our study is inspired by quantifying generalization in RL [11], which provides a random ϵ -greedy action that has been used to encourage exploration and prevent excessive over-fitting. A prioritized replay memory [13] has been exploited in a DQN to replay important transitions. A parameter space in the noise [14] of a neural network leads to a more abundant set of exploratory behaviors. These recent studies have shown how HER [8] can improve random ϵ -greedy actions. Therefore, we consider enhancing multiple random ϵ -greedy experience buffers in deep RL for more near-perfect explorations for the original goal. We demonstrate the off-policy learning benefit from our method through an experimental comparison of DQN [9,10] and DDPG [2,16] in high-dimensional discrete action environments as well as continuous control tasks. Our idea is that after experiencing some episodes, we store them in the replay buffers R_1 and R_2 . We can replay one trajectory with only one goal, assuming that we exploit an off-policy RL like DQN [9,10] and DDPG [2,16]. While the procedure passes through multiple random

ϵ -buffers, the balance between variance and bias of the policy can be solid and strong due to the training of the deep neural network.

Without using a complicated shaping reward and knowledge environment, we attempt to achieve a near-perfect global optimal goal. This idea is also used in the HER [8] and ODE model [12]. We attempt to achieve better exploration, which can balance computational speed and global optimization. If we use both a target policy and an alternative policy, we can attain the previously achieved global optimization. Subsequently, we consider the computational speed while maintaining generalization. HER [8] stores each transition in the replay buffer not only with the original goal, but also with a subset of other goals. However, we increase the number of replay buffers, which are a minimum of two. Therefore, we can replay each trajectory with two or more buffers, assuming that we use an off-policy RL algorithm, such as DQN [9,10] or DDPG [2,16], for a complete symmetric environment. Similar to HER [8], our proposed model can learn with extremely sparse rewards, and performs better in our experiments. These results indicate the practical challenges with reward shaping, such as HER [8]. Moreover, we can acquire the results of the effects of memory size from ODEs [12]. The selection of memory size does not have any effect on learning behavior; there is a trade-off between an over-fitting and the weight update [12]. If the memory size is smaller, the process of learning is probably over-fitted. Similarly, if the memory is bigger, the over-fitting effect is alleviated. However, it is not easy to deal with the balance between over-fitting and weight update [11,12]. With prioritized replay [13], if the memory size is small, especially if the mini-batch is small, then the trade-off between the over-fitting and weight update is more consequential. Therefore, we consider enhancing the characteristic feature of model-free RL, i.e., increasing the number of random ϵ -buffers. In terms of policy optimization, the policy $\pi_\theta(s,a)$ is explicit, and for performance optimization, the agent follows the policy $\pi_\theta(s,a)$ directly. Therefore, we call it on-policy optimization. However, Q-learning is based on off-policy, because it can learn about an optimal policy with respect to $Q(s,a)$, such as $\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$, and follow an exploratory policy [1]. Policy optimization [2,16] is more stable than Q-learning [9,10]; however, Q-learning [9,10] is more sample-efficient than policy optimization [2,16]. Therefore, we follow the lead to strengthen the advances in model-free RL. DDPG [16] is actually based on on-policy [1]. However, it can create a good trade-off between policy optimization [2,16] and Q-learning [9,10]. Algorithms 3 and 4 display the DQN and DDPG algorithm, respectively, with multiple random ϵ -buffers.

Algorithm 3 Deep Q-learning with multiple random ϵ -buffers

```

Initialize replay memory  $R_1$  and  $R_2$ 
Initialize Q-function with random weights
for episode = 1,  $M$  do
    Initialize sequence  $s_1$  and  $\phi_1 = \phi(s_1)$ 
    for t = 1,  $T$  do
        With probability  $\epsilon$ , select a random action  $\alpha_t$ 
        Or select  $\alpha_t = \max_{\alpha} Q^*((s_t), \alpha; \theta)$ 
        Execute action  $\alpha_t$  in emulator and observe reward  $r_t$ 
        Set  $s_{t+1} = s_t, \alpha_t$  and  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, \alpha_t, r_t, \phi_{t+1})$  in  $R_1$ 
        //standard experience buffer
        Store transition  $(\phi_t, \alpha_t, r_t, \phi_{t+1})$  in  $R_2$ 
        //second experience buffer
        Sample random mini-batch of transitions  $(\phi_j, \alpha_j, r_j, \phi_{j+1})$  from either  $R_1$  or  $R_2$ 
        Set  $y_j = r_j$ 
        Or  $y_j = r_j + \gamma \max_{\alpha'} Q^*(\phi_{j+1}, \alpha'; \theta)$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, \alpha_j; \theta))^2$ 
    end for
    Sample random mini-batch of transitions from either  $R_1$  or  $R_2$ 
end for

```

Algorithm 4 Deep Deterministic Policy Gradient with multiple random ε -buffers

```

Initialize critic  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ 
Initialize target  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$ 
Initialize replay memory  $R_1$  and  $R_2$ 
for episode = 1,  $M$  do
    Initialize observation  $s_1$  and a random process  $N$  for action exploration
    for  $t = 1, T$  do
        Select action  $\alpha_t = \mu(s_t|\theta^\mu) + N_t$  by the current policy and exploration noise
        Execute action  $\alpha_t$  and observe reward  $r_t$  and new state  $s_{t+1}$ 
        Store transition  $(s_t, \alpha_t, r_t, s_{t+1})$  in  $R_1$ 
        //standard experience buffer
        Store transition  $(s_t, \alpha_t, r_t, s_{t+1})$  in  $R_2$ 
        //second experience buffer
        Sample a random mini-batch of transitions  $(s_j, \alpha_j, r_j, s_{j+1})$  from either  $R_1$  or  $R_2$ 
        Set  $y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'}))|\theta^{Q'}$ 
        Update critic by minimizing the loss:  $L = \frac{1}{n} \sum_j (y_j - Q(s_j, \alpha_j|\theta^Q))^2$ 
        Update the actor by the sampled policy gradient:
         $\nabla_{\theta^\mu} L \approx \frac{1}{n} \sum_j \nabla_{\alpha} Q(s, \alpha|\theta^Q)|_{s=s_j, \alpha=\alpha_j} \mu(s|\theta^\mu)|_{s=s_j}$ 
        Update the target:
         $\theta^{Q'} \leftarrow \tau \theta^Q + (1-\tau) \theta^{Q'}$ 
         $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1-\tau) \theta^{\mu'}$ 
    end for
    Sample a random mini-batch of transitions from either  $R_1$  or  $R_2$ 
end for

```

3.2. Algorithm Description

3.2.1. DQN with Multiple Random ε -Buffers

1. Initialize replay memories of both R_1 and R_2 and $Q(s,a)$, and initiate the process from a random state, s .
2. Initialize the sequence with the start state, s .
3. The agent learns the policy $\max_{\alpha} Q^*((s_t,a);\theta)$ with greed and follows another policy with probability ε .
4. For exploration, an experience composed of a tuple, such as (state s , action a , reward r , new state s'), is in both R_1 and R_2 .
5. Sample a random mini-batch of transitions from either R_1 or R_2 .
6. The weights for performing the gradient descent $(r_j + \gamma \max_{\alpha'} Q^*(\phi_{j+1}, \alpha') - Q(\phi_j, \alpha_j|\theta))$ for a target DQN with multiple random ε -buffers.
7. Steps 3–6 are repeated for training.
8. For the next episode, sample a random mini-batch of transitions from either R_1 or R_2 .
9. Steps 2–8 are repeated for training.

3.2.2. DDPG with Multiple Random ε -Buffers

1. Initialize critic deep network $Q(s, a|\theta^Q)$ and actor deep network $\mu(s|\theta^\mu)$.
2. Initialize replay memories in both R_1 and R_2 .
3. The agent selects the action according to the current policy, $\alpha_t = \mu(s_t|\theta^\mu) + N_t$.
4. For exploration, an experience composed of a tuple, such as (state s , action a , reward r , new state s'), is in both R_1 and R_2 .
5. A mini-batch of transitions is randomly sampled from either R_1 or R_2 .
6. The critic network is updated by the loss function $L = 1/n \sum_j (r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'}))|\theta^{Q'}) - Q(s_j, \alpha_j|\theta^Q))^2$.

7. The actor network is updated by the policy gradient descent $\nabla_{\theta} \mu I \approx 1/n \sum_j \nabla_{\alpha} Q(s|\theta^Q)|_{s=s_j}, \alpha = \alpha_{\mu}(s_j) \nabla_{\theta} \mu(s|\theta^{\mu})|_{s_j}$ for a target DDPG with multiple random ε -buffers.
8. Update the target network $\theta^Q \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$ and $\theta^{\mu} \leftarrow \tau \theta^{\mu} + (1 - \tau) \theta^{\mu'}$.
9. Steps 3–8 are repeated for training.
10. For the next episode, a mini-batch of transitions is randomly sampled from either R_1 or R_2 .
11. Steps 2–10 are repeated for training.

4. Evaluation and Results

We consider OpenAI Gym [19] for our proposed algorithms, i.e., DQN and DDPG with multiple random ε -buffers. To enhance multiple random ε -greedy experience buffers, we exploit classic control environments in OpenAI Gym, such as CartPole-v0 [20,21], MountainCar-v0 [22,23], and Pendulum-v0 [24,25]. With similar approaches [9,10], we exploit a method, known as experience replay. Moreover, in our proposed method, Q-learning stores past experiences at each time step, $e_t = (s_t, a_t, r_t, s_{t+1})$, in multiple random ε -buffers, R_1 and R_2 . Our emulators from OpenAI Gym [19] can apply mini-batch updates into R_1 and R_2 . After the multiple random experience replay memories, the agent's actions of the emulator follow ε -greedy policy. In terms of DQN with multiple random ε -buffers, for enhanced exploration, we also follow the theory of the target Q-network of breakthrough research in [9,10]. The Q-learning agent calculates the TD error with the current estimated Q-value [9,10]. The optimized action-value function follows a significant identity, known as the Bellman equation [9,10]. In accordance with this equation, the TD target is the reward based on an action in the state plus the discounted highest Q-value for the next state [9,10]. In terms of DDPG with multiple random ε -buffers, for enhanced exploration, we also follow the theory of the critic and actor networks in [2,16]. Because the Q-learning of DDPG can exploit the deterministic policy, $\arg\max_a Q$, our proposed method can follow the off-policy [1].

For DQN with multiple random ε -buffers, we exploit CartPole-V0 [20,21] and MountainCar-V0 [22,23]. In CartPole-V0 [20,21], a pole is attached with an unactuated joint to a cart moving along a frictionless track. It is controlled by forcing +1 or −1 to the cart. The pole starts upright, and the goal is to prevent it from falling over. A + 1 reward is given to every time step in which the pole remains upright. The episode stops when the pole is more than 15° from the vertical direction or the cart moves more than 2.4 units from the center. In our simulation, CartPole-V0 defines “solving” as if the average reward is more than 490 or equal to 500 over 10 consecutive runs. The agent of CartPole-V0 receives −100 reward if it falls over prior to the max-length of the episode [21]. In MountainCar-V0, a car is on a one-dimensional track, positioned between two mountains, with the goal of driving up the mountain on the right. The engine of the car is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back-and-forth to reach the other side of the mountain. MountainCar-V0 defines “solving” as obtaining an average reward of −110 over 100 consecutive trails [23]. In Pendulum-V0, the inverted pendulum starts from a random position, and the goal is to swing it upwards so that it stays upright. Pendulum-V0 is an unsolved environment, which means that it does not have a special reward threshold at which it is considered solved [24,25].

Our proposed DQN and DDPG with multiple random ε -buffers were implemented using TensorFlow [26] and Keras [27] in OpenAI Gym [19].

4.1. CartPole-V0

For DQN with multiple random ε -buffers, we follow similar previous studies [21]. In Figure 1a, there is a CartPole-V0 [20] by OpenAI Gym [19]. Thus, GAMMA = 0.95, LEARNING – RATE = 0.001, MEMORY – SIZE = 10,000, BATCH – SIZE = 64, EXPLORATION – MAXIMUM = 1.0, EXPLORATION – MINIMUM = 0.01, and EXPLORATION – DECAY = 0.995 are the same as with the previous DQN studies [21] because of a fairness comparison.

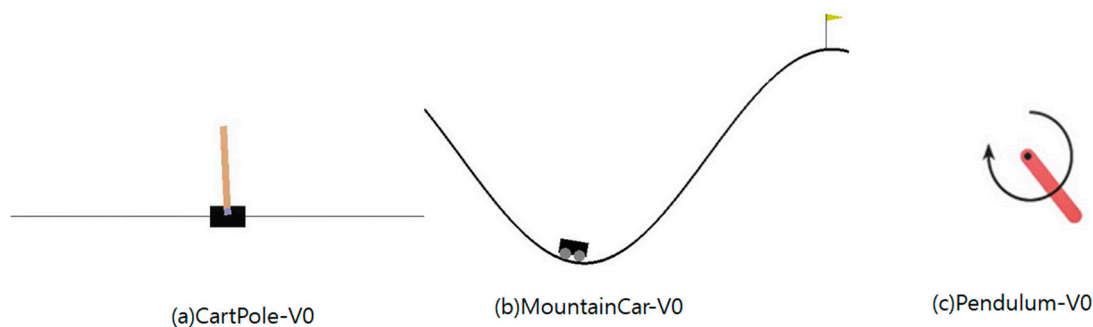


Figure 1. (a) CartPole-V0 [20], (b) MountainCar-V0 [22], and (c) Pendulum-V0 [24].

In the Cart-Pole-V0 environment, there are four observations and two discrete actions; see Figure 2. Furthermore, there are three episode terminations: the pole angle is more than -12° or $+12^\circ$, the cart position is more than -2.4 or $+2.4$, and the episode length is greater than 200 [20].

(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Cart Position	-2.4	+2.4
1	Cart Velocity	-Inf	+Inf
2	Pole Angle	$\sim -41.8^\circ$	$\sim +41.8^\circ$
3	Pole Velocity At Tip	-Inf	+Inf

(b) Actions

Number	Action
0	Push Cart to the Left
1	Push Cart to the Right

Figure 2. (a) The environment of CartPole-V0 and (b) the actions taken by the agent of CartPole-V0 [20].

However, the requirements from the implementation research can be considered in order to obtain better solutions [21]. Q-learning receives -100 as the reward when it falls before the maximum length of the episode is reached. Moreover, if the average reward is more than 490 or equal to 500 over 10 consecutive runs, Q-learning is terminated, even before it attains the maximum length of the episode [21].

Figures 3–5 display most of the results for the average, worst, and best cases, respectively. A DQN with multiple random ϵ -buffers can yield better results than one random buffer. In each of the average, worst, and best cases, our proposed method reaches the maximum reward earlier than DQN with one random buffer. Therefore, we can simulate identical results in real-time online for information security in Big Data. If our proposed method is used in training in real-time online in deep RL, as in a recent study [3–7], we acquire better results. Moreover, we can attempt a different deep neural network and hyper-parameter settings to demonstrate that our proposed model can enhance the exploration without specific domain information. We are convinced that we can enhance the behavior policy. Therefore, our proposed method can advance the exploration performance of ϵ -greedy, because multiple mini-batches do not follow a local minimum [9,10].

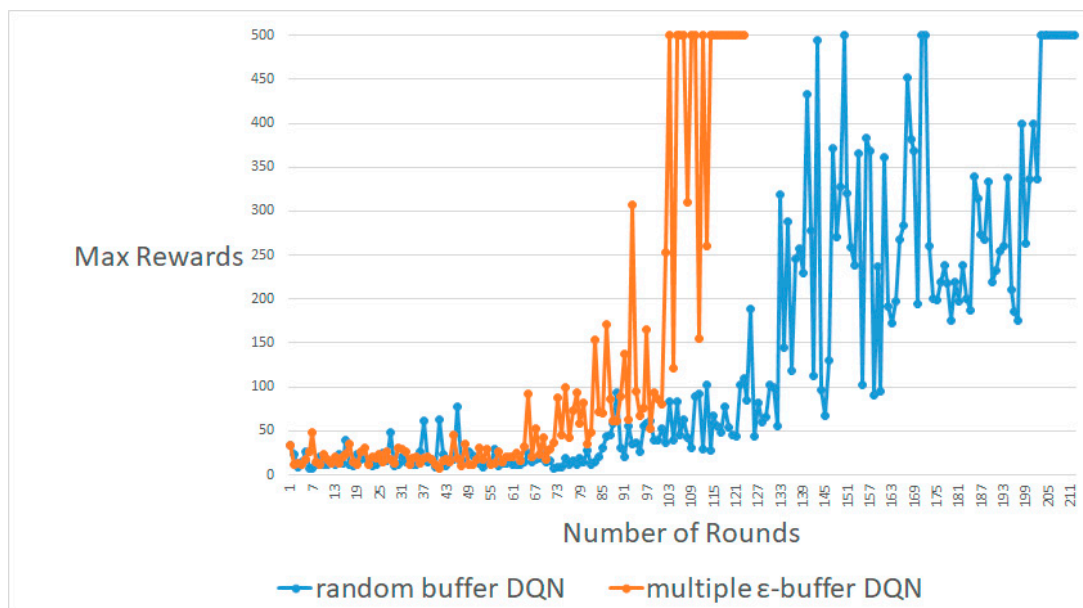


Figure 3. An average result between DQN with random buffers and DQN with multiple random ϵ -buffers in CartPole-V0.

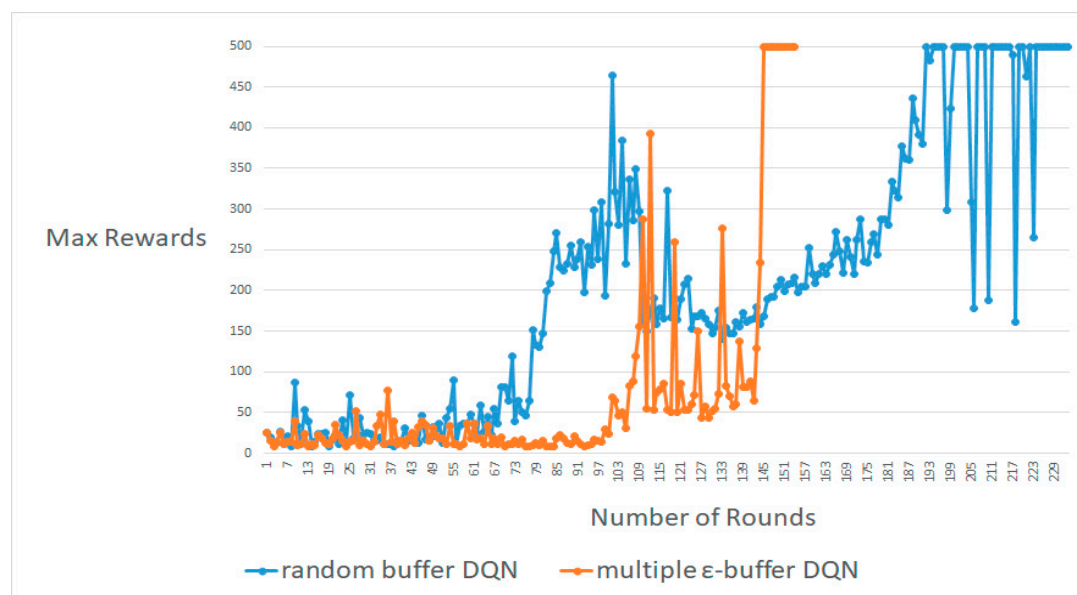


Figure 4. One of the best results between DQN with random buffers and DQN with multiple random ϵ -buffers in CartPole-V0.

4.2. MountainCar-V0

For DQN with multiple random ϵ -buffers, we follow similar previous studies [23]. Figure 5b shows a MountainCar-V0 [22] in OpenAI Gym [19]. Thus, GAMMA = 0.95, LEARNING – RATE = 0.001, MEMORY – SIZE = 100,000, BATCH – SIZE = 64, EXPLORATION – MAXIMUM = 1.0, EXPLORATION – MINIMUM = 0.01, and EXPLORATION – DECAY = 0.995 are the same as with previous DQN studies [23] because of a fairness comparison. In the MountainCar-V0 environment, there are two observations and three discrete actions, in Figure 6. Additionally, there are two episode terminations: when it reaches 0.5 position of the flag or if 200 iterations are reached. A penalty of 1 unit is applied for each move, including doing nothing. Our method limits the maximum number of times to 10,000 and 60 episodes by following the previous implementation of “Car has reached the goal” in each round [23].

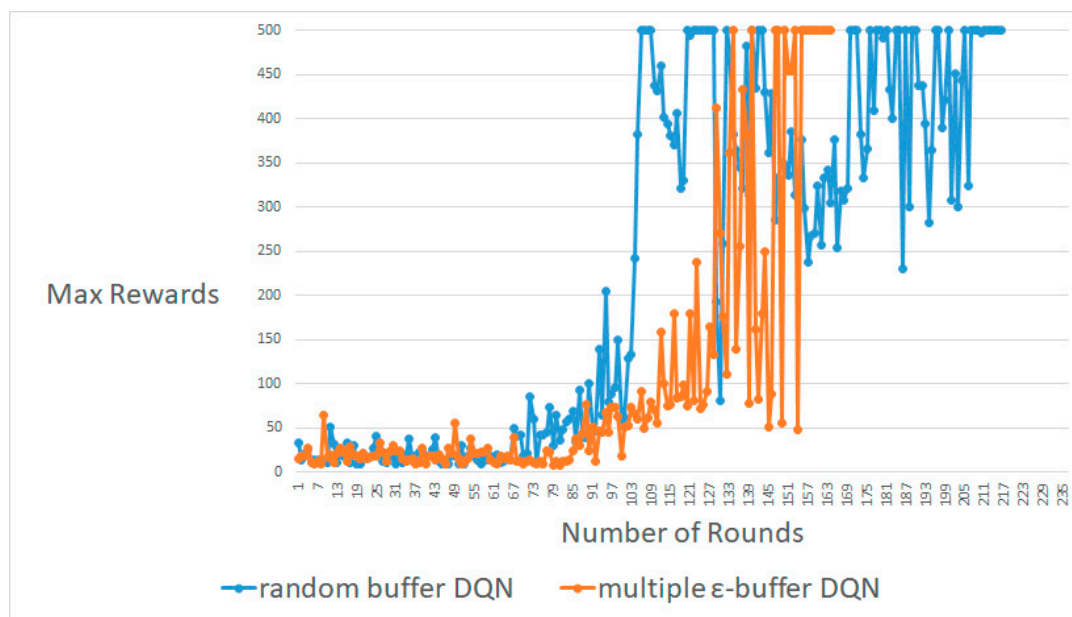


Figure 5. One of the worst results between DQN with random buffers and DQN with multiple random ϵ -buffers in CartPole-V0.

(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Position	-1.2	+0.6
1	Velocity	-0.07	+0.07

(b) Actions

Number	Action
0	Push Left
1	No Push
2	Push Right

Figure 6. (a) The environment of MountainCar-V0 and (b) the actions taken by the agent of MountainCar-V0 [22].

Figures 7–9 display the maximum results for the number of rounds, from approximately 100 to 300. A DQN with multiple random ϵ -buffers can yield better results compared to when one random buffer is used. In each case, for around 100, 200 and 300 rounds, our proposed method shows that the number of “Car has reached the goal” [23] is higher than that obtained by DQN with one random buffer. We guarantee similar results for deep learning car simulations online, such as real-time learning with a normal neural network. Moreover, we can expect a real-time system with live training for real-time safeties. Similarly, for the experiment on Cart-Pole, we exploit the behavior policy to implement better exploration with multiple random ϵ -buffers so as not to follow a local minimum [9,10].

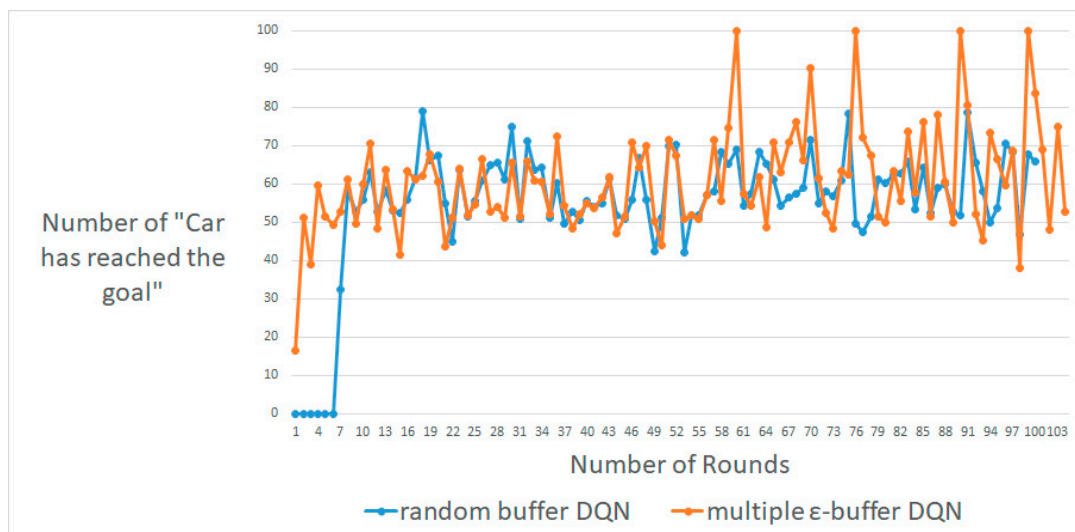


Figure 7. Maximum number of rounds, i.e., around 100, between DQN with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

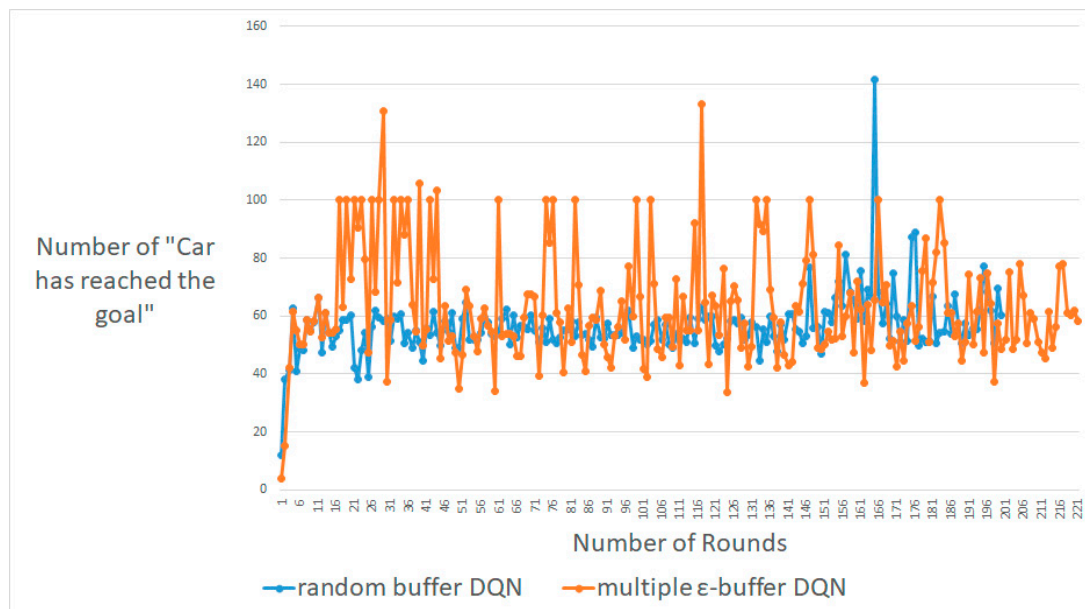


Figure 8. Maximum number of rounds, i.e., around 200, between DQN with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

4.3. Pendulum-V0

For DDPG with multiple random ϵ -buffers, we follow similar previous studies [25]. Figure 5c shows a Pendulum-V0 [24] by OpenAI Gym [19]. Thus, LEARNING – RATE = 0.001, MEMORY – SIZE = 2000, BATCH – SIZE = 32, EXPLORATION – MAXIMUM = 1.0, EXPLORATION – MINIMUM = 0.01, and EXPLORATION – DECAY = 1/2000 are the same as with the previous DQN studies [25] because of a fairness comparison. In the pendulum environment, there are three observations and one discrete action; see Figure 10. Additionally, there is a precise equation for the reward: $-(\theta^2 + 0.1 \times \dot{\theta}^2 + 0.001 \times \text{action}^2)$, where θ is normalized between $-\pi$ and $+\pi$. 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 [25]. The goal is to remain at zero angle (vertical) with the least rotational velocity and least effort. There are no episode terminations, adding a maximum number of steps [24,25].

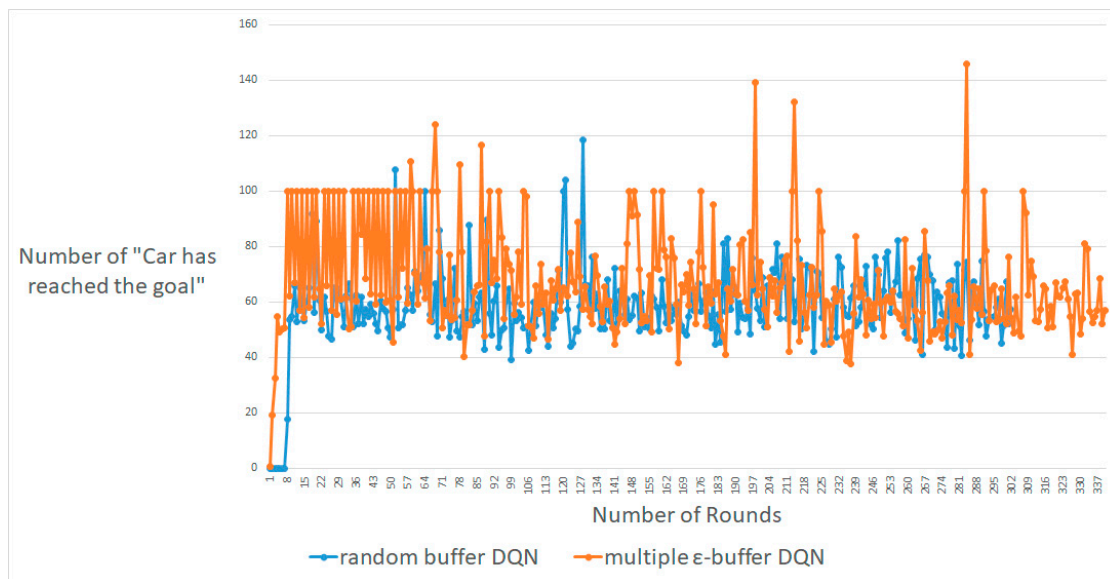


Figure 9. The maximum number of rounds, i.e., around 300, between DQN with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Cos(theta)	-1.0	+1.0
1	Sin(theta)	-1.0	+1.0
2	Theta dot	-0.8	+0.8

(b) Actions

Number	Action	Minimum	Maximum
0	Joint Effort	-2.0	+2.0

Figure 10. (a) The environment of MountainCar-V0 and (b) the actions taken by the agent of Pendulum-V0 [24].

Figures 11–13 exhibit the maximum results for the number of rounds, ranging from approximately 100–300. A DDPG with multiple random ϵ -buffers can yield a small improvement compared to that with one random buffer. In each case, for around 100, 200 and 300 rounds, our proposed method shows higher rewards than DDPG [16] with one random buffer. However, unlike the results of CartPole-V0 [20] and MountainCar-V0 [22], DDPG with multiple random ϵ -buffers cannot obtain a significant result compared with the cases of DQN. We consider this from the viewpoint of the on-policy. Fundamentally, a DDPG is based on the on-policy. Moreover, there is an issue about interpolating between policy optimization [2,16] and Q-learning [9,10]. The multiple random ϵ -buffers are actually based on Q-learning.

To train an agent, there are two major algorithms in model-free RL: policy optimization and Q-learning, which can follow the Bellman equation. A Q-learning agent selects an action to maximize the Q-value function based on the data at some point in the environment, regardless of the policy, known as the off-policy. Therefore, explorations on sample efficiency are extremely important to maximize the objective Q-value function. The representative algorithm in the off-policy is DQN. However, in terms of policy optimization, for better performance, an agent follows the recent deterministic policy and exploits a gradient ascent of the deterministic policy, thereby optimizing the parameters for the deterministic policy, known as the on-policy. Therefore, the update of the deterministic policy is

extremely important for the exploration of the sample data. DDPG trains the agent by developing both the deterministic policy and the Q-value function. However, it is difficult to obtain both principled policy optimization and efficient Q-learning. That is why we cannot achieve better results in terms of DDPG. Therefore, we developed “Tradeoff Policy Optimization and Q-Learning” based on “Data vs. Policy” for better DDPG results. Thus, superior results can be obtained if we exploit a policy optimization method, based on the on-policy. This is another issue for future research.

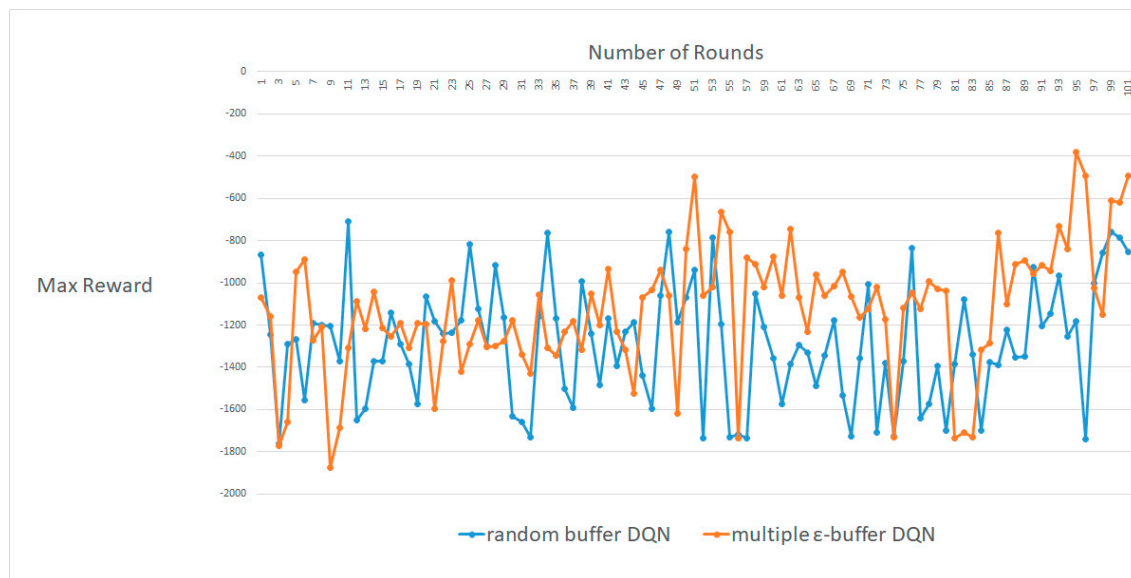


Figure 11. Maximum number of rounds, i.e., around 100, between DDPG with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

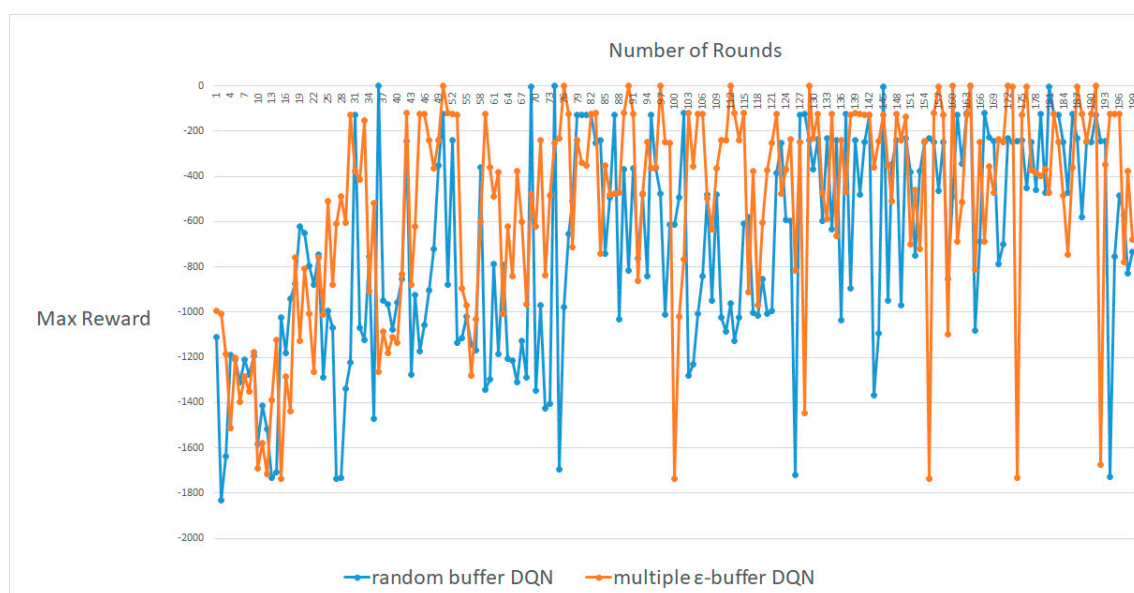


Figure 12. Maximum number of rounds, i.e., around 200, between DDPG with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

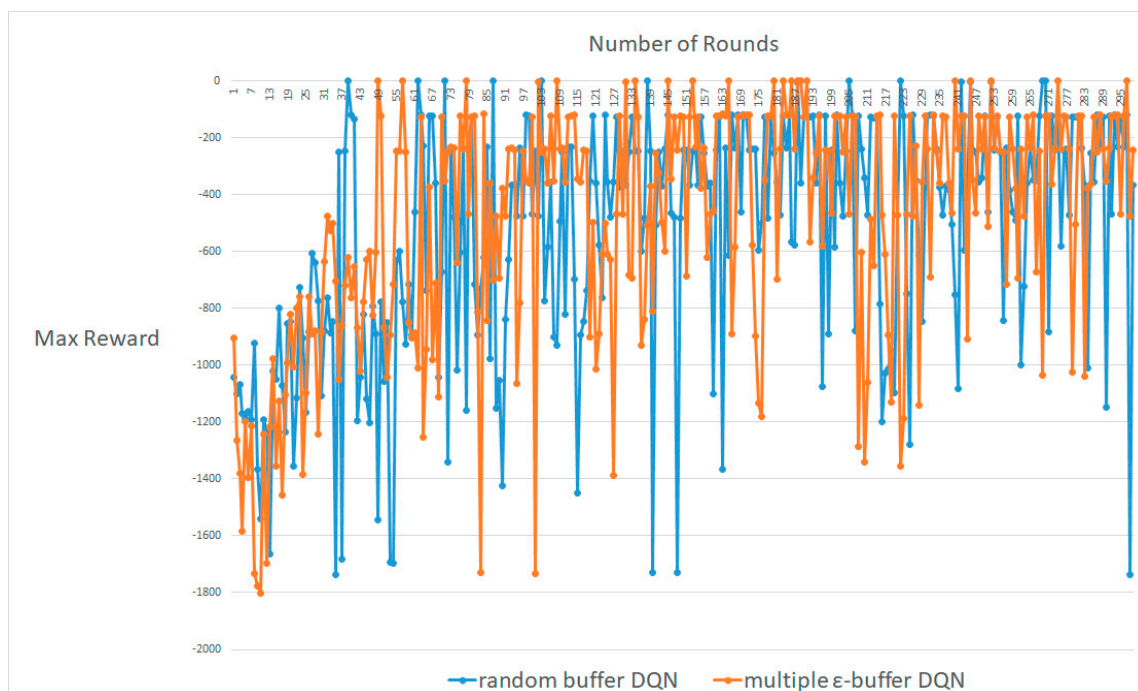


Figure 13. Maximum number of rounds, i.e., around 300, between DDPG with random buffers and DQN with multiple random ϵ -buffers in MountainCar-V0.

5. Conclusions

We proposed multiple ϵ -greedy experience buffers in off-policy deep RL to enhance the exploration for superior and near-perfect generalization. We considered strengthening the advantages of explorations of model-free deep RL. We exploited multiple random ϵ -buffers for one original goal, and emulated the environment of OpenAI Gym to achieve better results in environments such as CartPole-V0, MountCar-V0, and Pendulum-V0. Our results demonstrated that the off-policy method was beneficial through an experimental comparison of DQN and DDPG based on on-policy. The proposed model is compatible with discrete actions as well as continuous control, symmetrically. Therefore, we can expect better prediction accuracy in real-time online learning, as in the detection of network intrusions through the verification of whether the network is “normal or anomalous.” Although our results are superior to those obtained by normal DQN with one random buffer, there is still scope for improvement, particularly for balancing policy optimization and Q-learning. The multiple random ϵ -buffers are fundamentally based on Q-learning. Therefore, the trade-offs between policy optimization and Q-learning should be solid and strong. Because of the deep neural network in RL, better results can be acquired in terms of DDPG, which is based on the on-policy of policy optimization. However, we can obtain better results if we develop another method based on policy optimization beyond the multiple random ϵ -buffers. The comparisons conducted in this study reveal that our model displays improvements in terms of exploration over the deep RL algorithms with Q-learning systems.

Author Contributions: Conceptualization, C.K. and J.P.; methodology, C.K.; writing—original draft preparation, C.K.; writing—review and editing, J.P.; project administration, J.P.; funding acquisition, J.P.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF2017R1D1A1B03035833).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998; Volume 1.

2. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv* **2018**, arXiv:1801.01290.
3. Kim, C.; Park, J. Designing online network intrusion detection using deep auto-encoder Q-learning. *Comput. Electr. Eng.* **2019**, *79*, 106460. [[CrossRef](#)]
4. Park, J.; Salim, M.M.; Jo, J.; Sicato, J.C.S.; Rathore, S.; Park, J.H. CIoT-Net: A scalable cognitive IoT based smart city network architecture. *Hum. Cent. Comput. Inf. Sci.* **2019**, *9*, 29. [[CrossRef](#)]
5. Sun, Y.; Tan, W. A trust-aware task allocation method using deep q-learning for uncertain mobile crowdsourcing. *Hum. Cent. Comput. Inf. Sci.* **2019**, *9*, 25. [[CrossRef](#)]
6. Kwon, B.-W.; Sharma, P.K.; Park, J.-H. CCTV-Based Multi-Factor Authentication System. *J. Inf. Process. Syst. JIPS* **2019**, *15*, 904–919. [[CrossRef](#)]
7. Srilakshmi, N.; Sangaiah, A.K. Selection of Machine Learning Techniques for Network Lifetime Parameters and Synchronization Issues in Wireless Networks. *Inf. Process. Syst. JIPS* **2019**, *15*, 833–852. [[CrossRef](#)]
8. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight Experience Replay. *arXiv* **2017**, arXiv:1707.01495.
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
10. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Driessche, G.V.D.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. Mastering the game of go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [[CrossRef](#)] [[PubMed](#)]
11. Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; Schulman, J. Quantifying Generalization in Reinforcement Learning. *arXiv* **2019**, arXiv:1812.02341.
12. Liu, R.; Zou, J. The Effects of Memory Replay in Reinforcement Learning. *arXiv* **2017**, arXiv:1710.06574.
13. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized Experience Replay. *arXiv* **2016**, arXiv:1511.05952.
14. Plappert, M.; Houthoofd, R.; Dhariwal, P.; Sidor, S.; Chen, R.; Chen, X.; Asfour, T.; Abbeel, P.; Andrychowicz, M. Parameter Space Noise for Exploration. *arXiv* **2018**, arXiv:1706.01905.
15. OpenReview.net. Available online: <https://openreview.net/forum?id=ByBAI2eAZ> (accessed on 16 February 2018).
16. Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2016**, arXiv:1509.02971.
17. FreeCodeCamp. Available online: <https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682> (accessed on 5 July 2018).
18. RL—DQN Deep Q-network. Available online: https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4 (accessed on 17 July 2018).
19. OpenAI Gym. Available online: <https://gym.openai.com> (accessed on 28 May 2016).
20. Cart-Pole-V0. Available online: <https://github.com/openai/gym/wiki/Cart-Pole-v0> (accessed on 24 June 2019).
21. Cart-Pole-DQN. Available online: https://github.com/rlcode/reinforcement-learning-kr/blob/master/2-cartpole/1-dqn/cartpole_dqn.py (accessed on 8 July 2017).
22. MountainCar-V0. Available online: <https://github.com/openai/gym/wiki/MountainCar-v0> (accessed on 4 May 2019).
23. MountainCar-V0-DQN. Available online: <https://github.com/shivaverma/OpenAIGym/blob/master/mountain-car/MountainCar-v0.py> (accessed on 2 April 2019).
24. Pendulum-V0. Available online: <https://github.com/openai/gym/wiki/Pendulum-v0> (accessed on 31 May 2019).
25. Pendulum-V0-DDPG. Available online: https://github.com/openai/gym/blob/master/gym/envs/classic_control/pendulum.py (accessed on 26 October 2019).
26. Tensorflow. Available online: <https://github.com/tensorflow/tensorflow> (accessed on 31 October 2019).
27. Keras Documentation. Available online: <https://keras.io/> (accessed on 14 October 2019).

