Enhancing Deep Deterministic Policy Gradients on Continuous Control Tasks with Decoupled Prioritized Experience Replay

Dogan C. Cicek, Aykut Koc, *Senior Member, IEEE*, and Suleyman S. Kozat, *Senior Member, IEEE*

***Abstract*— Experience replay methods in the literature provide identical batches of transitions to the Actor and the Critic. However, the learning principles of these two cascaded components of deep deterministic policy gradients algorithms contain dissimilarities in terms of their updating strategies. We decouple the training of the Actor and the Critic of the deep deterministic policy gradient algorithms in terms of the batches of transitions that they use during the training of the networks. Our approach, Decoupled Prioritized Experience Replay, DPER, enables the agent to use independently sampled batches of transition for the Actor and the Critic of the Deep Deterministic Policy Gradient Algorithms. We combine our algorithm with the current state-of-the-art Deep Deterministic Policy Gradients algorithm, Twin Delayed DDPG(TD3), and evaluate it on continuous control tasks. Also, DPER is usable for every off-policy deep deterministic reinforcement learning algorithm that works on continuous control tasks. DPER outperforms the conventional experience replay mechanisms without adding a significant amount of computational complexity to most of the continuous control tasks.**

***Index Terms*— reinforcement learning, machine learning for robot control, learning from experience, prioritized sampling, continuous control**

I. Introduction

Reinforcement learning shows remarkable advancement when combined with deep learning. Deep reinforcement learning started to become more popular after it was used to tackle human control tasks on ATARI [1]. Also, deep reinforcement learning approaches solve distinct continuous control tasks such as human-robot collaboration [2], motion planning [3], path planning [4], telerobotics and teleoperation [5], vehicle decision making [6],and endoscopic capsule robot navigation [7], with remarkable success.

Deep reinforcement learning algorithms solve tasks with sufficiently big state and action spaces since it allows the agent to generalize the Q-values of each state-action pairs and yield a parameterized policy. Deep reinforcement learning agents train their policy by using the observation and reward information they receive [8]. However, observation information collected by the agent during consecutive time steps is highly correlated, which violates proper neural network training. The experience replay mechanism tackles the given problem by enabling the agent to reuse its past experiences, in other words, using previous transitions that it collected during the training [9]. That solution breaks the temporal correlation between transitions that are used to update the parameters of the Actor and the Critic. Experience replay mechanism enhances the performance of the deep reinforcement learning agents, provides them a sustainable return improvement and a robust policy during the learning process [10].

The most primitive variant of the experience replay method, Vanilla Experience Replay, samples transition from the replay buffer uniformly [9]. This approach directly assumes that each transition stored to the replay buffer is equally beneficial for the training process of the agent. On the contrary, the works show that the strategy the agent uses how to choose the transition it collects is heavily affecting the performance of the agent [11], [12]. Developing better experience replay mechanisms is an active research field, and there are studies that surpass the performance of the Vanilla Experience Replay on continuous control tasks [13], [14], [15], [16].

In this work, we propose a novel approach to prioritization of the collected transitions, Decoupled Prioritized Experience Replay, DPER. In our work, we decouple the training of the Actor and the Critic of the Deep Deterministic Policy Gradient Algorithms in terms of the batches of transitions that they use during the learning process. Our algorithm is usable for every off-policy deep deterministic reinforcement learning algorithm that works on continuous control tasks. We aim to show that it is possible to improve the performance of a deep reinforcement learning algorithm that has two cascaded neural structures, the Actor and the Critic, by updating them with different batches of transitions. To support our aim, we use two different Experience Replay methods in terms of their prioritization strategy. We show that a combination of two different Experience Replay algorithms can yield better results in particular continuous control tasks.

We evaluate our algorithm, DPER, by coupling it with the TD3 algorithm [17]. We collate DPER with Vanilla Experience Replay, KLPER, and Prioritized Experience Replay. We use OpenAI gym and MuJoCo learning environments which consist of a wide range of robot control tasks to test the performance of our proposed algorithm [18], [19].

We summarize our main contributions in this paper as follows:

- DPER is the first work that decouples the Actor and the Critic training by using a different batch of transitions,
- Develop DPER to make the Critic learn through transitions that yield more temporal difference error, while the Actor uses more on-policy batches of transition to update its parameters.
- We show that a combination of two experience replay methods can outperform the cases that they used sepa-

[1]The authors are with Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey (e-mail: cicek@ee.bilkent.edu.tr, aykut.koc@bilkent.edu.tr, kozat@ee.bilkent.edu.tr)

rately.
- We study and analyze the metrics used for the prioritization process of the transitions for both the Actor and the Critic training.
- We demonstrate that DPER brings significant performance improvements to the state-of-the-art deep deterministic policy gradients algorithm TD3 on continuous control tasks.

II. Background

In this section, we give a brief introduction to reinforcement learning and summarize two deep deterministic policy gradients algorithms designed for solving continuous control tasks. We also briefly cover the current experience replay methods in the literature.

A. Reinforcement Learning

A reinforcement learning agent optimizes its policy to obtain the maximum amount of return by consecutive trial and error. Reinforcement Learning tasks are defined as Markov Decision Processes (MDP) [8]. During the learning process, the agent receives state information $s \in \mathcal{S}$. After observing the state information, the agent chooses its action from the action space that the reinforcement learning problem have, $a_t \in \mathcal{A}$, with respect to its policy $a_t \sim \pi(a|s_t)$. Then, the agent applies the selected to the environment and receives the reward, r , and the next state information, $s' \in \mathcal{S}$ from the environment. The environment gives the reward and the next state to the agent according to the probability distribution $P(s', r|s, a)$.

The collected information throughout the above cycle is defined as a transition, (s, a, r, s'). Agents that work with experience replay mechanisms keep these transitions in their replay memory and use them to optimize the return function:

$$R_t = \sum_{i=t}^{T} \tau^{i-t} r(s_i, a_i), \tag{1}$$

where τ represents the discount factor.

B. Deep Deterministic Policy Gradient

One of the first deterministic deep reinforcement learning algorithms that tackle tasks with continuous action space is the Deep Deterministic Policy Gradients(DDPG) [20]. The DDPG algorithm has two cascaded deep neural network structures named the Critic and the Actor. The Critic estimates the Q-value of the given state-action pair as follows:

$$y = C(s, a|v), \tag{2}$$

where C is the Critic, v is the parameters of the Critic, and y is the Q-value of the given state-action pairs. The property of the Critic is that estimating Q-value by taking state and action information as the input enables the algorithm to work correctly on continuous action spaces. On the other hand, the Actor controls the agent's behavior with a parametric policy. So, it determines which action should be taken by the agent given the state information:

$$a = A(s|w), \tag{3}$$

where A is the Actor, w is the parameters of the Actor, and a is the action that is produced by the Actor. In addition to these aforementioned networks, the DDPG algorithm also has one more nested neural network structure named Target Network. This nested neural structure includes the Actor Target, which shares the same weights with the Actor at the beginning of the training. Also, the Critic Target has the same weights as the Critic at the start of the training. The Critic and the Actor updates are consecutive for deep deterministic policy gradients algorithms. In the conventional update method for these networks, first, a one step temporal difference error is calculated, and the loss function is defined with respect to this value:

$$\mathcal{L}(v) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}[(\mathcal{Y} - C(s, a|v)]^2, \tag{4}$$

where $\mathcal{B}$ is the experiences that collected by the agent, the target value for Critic is defined as:

$$\mathcal{Y} = r + C'(s', A(s'|w')|v'), \tag{5}$$

where C' is the Critic Target, A is the Actor Target, w', and v' are the weights of the A' and the C'. The output of the Critic can be assumed as the objective function of the agent, and it represents the discounted cumulative reward of the agent at a given state by following the policy that is parameterized by the Actor:

$$J(v, w) = \mathbb{E}_{s\sim\mathcal{B}}[C(s, a|v)|_{a=A(s|w)} A(s|w)], \tag{6}$$

Therefore, by taking the gradient of the $J(v, w)$ with respect to the Actor's weights, the Actor is updated:

$$\nabla_w J(v, w) = \mathbb{E}_{s\sim\mathcal{B}}[\nabla_a C(s, a|v)|_{a=A(s|w)} \nabla_w A(s|w)], \tag{7}$$

It has been shown that bootstrapping from the same neural network significantly harms the training process [21]. To remedy the given issue, the DDPG algorithm utilizes Target-Networks. The Actor Target and the Critic Target are softly updated by using the Actor and the Critic:

$$w' = \gamma w + (1 - \gamma)w', \quad v' = \gamma v + (1 - \gamma)v', \tag{8}$$

where γ is the rate of the soft update.

C. Twin Delayed DDPG

Studies suggest that even though the DDPG algorithm performs well on continuous control tasks, it suffers from an overestimation problem [17], [20]. Overestimation problem is defined as estimating the Q-value of a given state-action pair, (s, a), while following a policy, w, higher than it should be:

$$C_\pi(s, a|v) > Q_\pi(s, a), \tag{9}$$

where C_π is the estimated Q-value of the given state-action pair while following the policy π, and Q_π is the true Q-value of the given state-action pair while following the policy π. Despite the presence of the Target Networks, in some cases, bootstrapping on overestimated state-action pairs dominantly accelerates the overestimation problem and cripples the learning process. Moreover, it may lead to the

agent instantly forgetting its policy in the middle of the training [12]. TD3 remedies this problem by adding one more Critic to the neural network structure [17]. TD3 algorithm formulates the target value for the Q-value of a given state-action pair as follows:

$$\mathcal{Y} = r + \tau \min_{i=1,2} C_i'(s', A'(s'|w')|v_i'), \tag{10}$$

and the authors call this procedure Clipped Double Q-Learning [17]. This advancement enables the TD3 to dominantly outperform the DDPG in continuous control tasks. The TD3 solves particular tasks that the DDPG cannot find any decent policy.

D. Experience Replay Mechanism

Neural Networks yield much better results when the training dataset includes mostly uncorrelated samples. However, in reinforcement learning, the agent receives state, and next state information is overlapped for successive transitions, i.e., the agent's next state, s', at time step t, is the same as the state, s, at the time step $t+1$. Then, training deep reinforcement learning is challenging since the agent updates its policy and value function using heavily correlated samples from the environment.

The experience replay mechanism remedies the given fact by adding a cyclic replay buffer to the algorithms. The first experience replay method stores the experiences that the agent collects while exploring the environment to the replay buffer as transitions. Then, the agent updates its parameters using the samples selected from the replay buffer to break the correlation between samples that using for the training at a time. The most primitive method, Vanilla Experience Replay, samples transitions from the replay uniformly.

Prioritized Experience Replay is one of the most well-known techniques used for assigning a different level of importance for the transitions [13]. PER increases the tendency to feed the value function of the deep reinforcement learning algorithm for the transitions collected while the agent encounters unexpected outcomes. PER measures the unexpectedness level of a transition by using temporal difference error as a proxy.

Transitions in the replay buffer can be sampled by following heuristic rules. Focused Experience Replay introduces an alternative probability distribution, half normal distribution, to increase the sampling probabilities of the recently added transitions to the replay buffer [22]. Hindsight Experience Replay, HER, is an experience reuse mechanism that provides sub-goals for the agent to divide the main task into smaller tasks through experience replay [23].

Experience Replay is also can be formulated as a learning problem that proceeds parallel with the training of the deep reinforcement learning agent [14]. By utilizing a parameterized experience reuse strategy, the Neural Experience Replay Sampler technique manages experience replay prioritizing [24]. This method gives scores that determine the importance level of the transitions when the corresponding transition is collected with respect to replay policy.

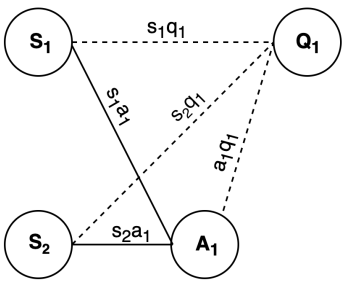


Fig. 1. Basic DDPG Neural Network structure

III. Motivation

In this section, we discuss the drawbacks that may be encountered by experience replay prioritization by following the procedure that the PER algorithm proposes. We also discuss the motivation behind the decoupling of the Actor and the Critic training by using different batches of transition for updating these networks. Lastly, we elaborate on the advantages of reducing off-policiness of a learning algorithm.

A. Advantages and Disadvantages of Prioritizing Experience Replay

Prioritized Experience Replay assigns sampling probabilities to the transitions. We remark that each transition's sampling probability is proportional to the temporal difference error that they produce:

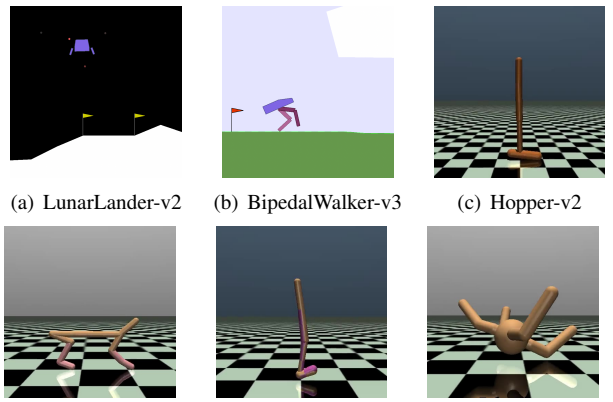
$$\delta = r(s, a, s') + \gamma C'\left(s', A'(s'|w')|v'\right) - C(s, a|v), \tag{11}$$

Intuitively, this procedure increases the tendency to train the Critic through the transitions that the Critic incorrectly estimates the Q-value that they yield. Then, it accelerates the learning process of the Critic. However, it is not the case when this idea is applied to the Actor update. To elaborate on this main drawback of Prioritized Experience Replay, Fig. 1 illustrates the main cascaded actor-critic network of a DDPG algorithm which is designed for a simple task that has two dimensions and one dimension in state and action spaces, respectively. Dashed and straight lines represent the weights of the Critic and the Actor, respectively. As we discuss the weight update sequence later, the DDPG and TD3 algorithms update their Critics first. Then, the Actors of these algorithms update their weights to improve the objective function. During the optimization step, the algorithms use the weights of the Critic that connect the action produced by the Actor to the Q-Value calculation. In our simple DDPG network illustration, in Fig. 1, these weights correspond to the weight $a1q1$. Therefore, the Actor can be updated correctly when the Critic optimization is decent for the given transition set. Conversely, Prioritized Experience Replay suggests transitions that the Critic produces a high temporal difference error to the Actor. Then, updating the Actor with this learning procedure significantly deteriorates the Actor optimization process.

B. Disadvantages of the Off-Policy Learning

An agent that uses a conventional reinforcement learning algorithm solely must cover all the states and actions that both state and action spaces have to yield a reasonable policy to solve the given problem. In this case, finding a decent policy becomes infeasible when the problem has big state and action spaces. Therefore, utilizing neural networks as approximation tools enables the agent to generalize the Q-values of the state-action pairs and plays a huge role in tackling the given type of problem.

Vanilla experience replay assumes each transition that the agent collects from the environment has the same level of importance for the training. So, at first glance, it is likely to come up with the idea that an experience replay mechanism outperforms the Vanilla Experience Replay. However, It is shown that prioritizing the process of the stored transition should be inspected carefully since it increases the off-policiness of the learning algorithm that the experience replay mechanism is coupled [12], [15]. Studies investigate how the changes in the Vanilla Experience Replay mechanism affect the learning process of the deep reinforcement learning agent [12], [13], [15]. One of them states that even the size of the cyclic experience replay buffer is crucial for proper learning for the agent since it indirectly determines the off-policiness level of the transitions that the agent receives during the training [15]. By altering the sample probabilities of the transitions, off-policy learning may be mitigated, and the improved algorithm performs better on learning tasks [12]. Reducing the off-policiness level of the learning algorithm is significantly important because the increased off-policiness may lead to divergent updates, which is highly undesirable.



(a) LunarLander-v2 (b) BipedalWalker-v3 (c) Hopper-v2

(d) HalfCheetah-v2 (e) Walker2d-v2 (f) Ant-v2

Fig. 2. OpenAI Gym [18] and MuJoCo Gym environment [19] tasks used for the experiments.

IV. Decoupled Prioritized Experience Replay

In this section, we provide the details of our algorithm, DPER. Also, we give more details on the batch selecting strategies of our algorithm for the Actor and the Critic.

A. Reducing the Off-Policiness of the Learning Algorithm

Every sample in the replay buffer is collected by the latest behavior policy that the agent follows while exploring the learning environment. Theoretically, one can store the policy that generates each transition by saving the all weights of the Actor. However, it is computationally infeasible since the Actor includes millions of weights, and the experience replay buffer stores thousands of transitions. Therefore, our algorithm assumes that each batch of transitions that are sampled from the replay buffer is generated by a stochastic policy. We refer to the policy that is most likely to produce a specific batch of transitions as the Transition Generator, and we denote it as λ. To define the policy that yields the batch of transitions, we use the agent's target policy as a proxy. Our algorithm feed forwards the states in the sampled batch from the replay buffer to the Actor. Then, based on the most current policy of the agent, the Actor outputs the actions that represent how the agent behaves in the specified states:

$$\hat{\boldsymbol{X}}^{b\times m} = A(\boldsymbol{S}^{b\times n}|w), \tag{12}$$

where b, A, m and n represent batch size, the Actor, action space dimension size and state space dimension size, respectively. To define the Transition Generator, DPER computes the difference between the actions chosen by the agent when the transition is stored and the actions the agent would take according to its latest policy:

$$\dot{\boldsymbol{X}}^{b\times m} := \hat{\boldsymbol{X}}^{b\times m} - \boldsymbol{X}^{b\times m}, \tag{13}$$

where $\boldsymbol{X}^{b\times m}$ is the action information kept in the sampled transitions, we remark that the Transition Generator cannot be tracked explicitly [16], unless storing all the Actor parameters at each timestep during the training. Therefore, we define it as a stochastic policy which is a probability distribution that covers the action space of the given task. We assign the mean of the mentioned probability distribution by measuring the deviation between the action information that is kept in the sampled transitions and the actions the agent yield on associated states:

$$\mu_{\lambda}^{1\times m} = \frac{1}{b}\sum_{i\in b} \dot{\boldsymbol{X}}_{ij}^{b\times m}, \tag{14}$$

where i and j represent each transition in the sampled transition and each degree of freedom of the action space, respectively, as we defined Transition Generator as a probability distribution, we determine its covariance as follows:

$$\Sigma_{\lambda}^{m\times m} = \frac{1}{b-1}\sum_{k\in b}(\dot{\boldsymbol{x}}_{k}^{1\times m} - \mu_{\lambda}^{1\times m})^{\top}(\dot{\boldsymbol{x}}_{k}^{1\times m} - \mu_{\lambda}^{1\times m}), \tag{15}$$

We represent the Transition Generator as a multivariate Normal distribution to maximize the entropy:

$$\lambda \sim N(\mu_{\lambda}^{1\times m}, \Sigma_{\lambda}^{m\times m}). \tag{16}$$

where λ is the Transition Generator of the given transitions.

We compare the most recent policy of the agent and batch generating policies of each batch indirectly by measuring the batch generating policies' deviation from the behavior policy

of the agent. We use KL divergence to quantify the deviation level. KL Scores of each batch are calculated as:

$$\eta = \mathrm{D}_{\mathrm{KL}}(N(\mu_\lambda, \Sigma_\lambda) \| N(0, \sigma\mathbb{I})), \tag{17}$$

where $\mathbb{I}$ represents the identity matrix. $N(0, \sigma\mathbb{I})$ represents the exploration noise that we add to yield the exploration policy of the agent. We assume that if the mean of the Transition Generator is equal to zero and its sample covariance matrix is a diagonal matrix such that its entries are equal to the exploration noise, our algorithm achieves updating the Actor parameters in an on-policy manner. However, it is not trivial to find a batch of transitions that ensures the aforementioned condition. Therefore, we limit the trial count, K, that our algorithm search for a perfect matching condition. In this case, our algorithm chooses the batch of transitions that minimizes η among a certain number of batches. By following this procedure, we force the Actor to be updated through more on-policy samples.

Algorithm 1 DPER

```
Set batch size b, policy delay parameter M
Set number of maximum batch trial K
Start experience memory B
for t = 1 to T do
  Observe s_t Choose action a_t ~ π_w(a|s_t) + ε
  Observe reward r_t and next state s'_t
  Store transition (s_t, a_t, r_t, s'_t) in B
  for i = 1 to b do
    Sample batch of transitions D_i
    Compute TD-error
    Update transition priority
  end for
  Update the Critic with D
  if k mod M then
    Choose K batch of transitions from B
    for n = 1 to N do
      Compute η_n for batch of transitions
    end for
    Select the batch, D, that yields minimum η
    Update the Actor with D
    Update the Actor and Critic Target
  end if
end for
```

B. Training the Actor and the Critic with Different Batch of Transitions

We choose the proportional Prioritized Experience Replay method as the transition sampling strategy of our algorithm's Critic training phase with changes [13]. Proportional Prioritizes Experience Replay mechanism adjusts sampling probabilities of each transition that is stored proportionally to the magnitude of the temporal difference error that they yield:

$$\psi = |r(s, a, s') + \gamma C'(s', a'|v') - C(s, a|v)|, \tag{18}$$

We denote the magnitude of the temporal difference error as ψ. In our algorithm, we assign a sampling probability to the transition right after it is gathered by the agent. Reassigning sampling probabilities of each transition after each time step is impractical when the replay buffer includes a substantial amount of transitions. Therefore, our algorithm does not change the sampling probabilities of the transitions until they are sampled from the replay buffer as the proportional Prioritized Experience Replay algorithm does. One main drawback of this procedure can occur at the beginning of the learning process. Assume that one transition that is stored to the buffer at the beginning of the replay outputs a minuscule temporal difference error. It would have a tiny sampling probability that prevents it from being resampled again, and it may occupy the replay buffer until being overridden by another transition. However, the importance level of this transition may change during the training in terms of the magnitude of the temporal difference error that it produces since the Critic and the Critic Target parameters change during the training. Therefore, to prevent our algorithm for such cases, we use the bias correction parameter, α and prioritize transitions as follows for the Critic training by following the process that the proportional Prioritized Experience Replay method proposes:

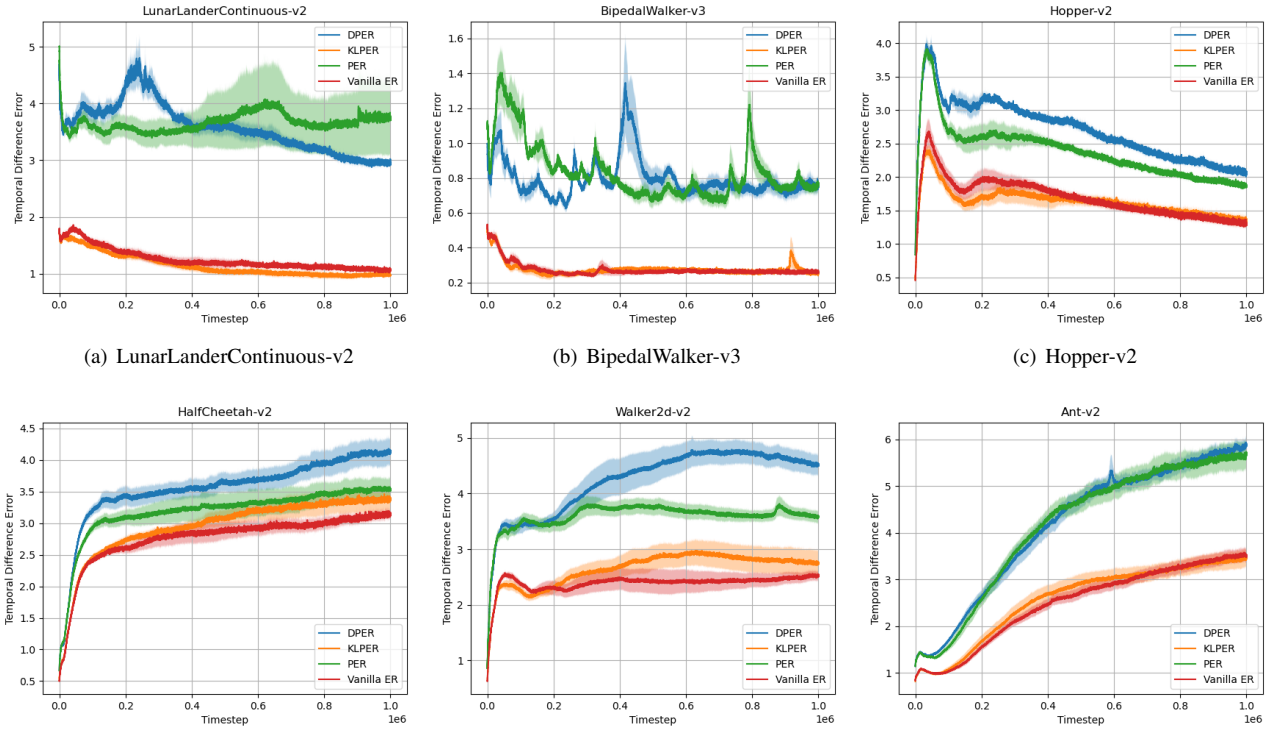
$$P(i) = \frac{\psi_i^\alpha}{\sum_k \psi_k^\alpha}, \tag{19}$$

where, $P(i)$ is the sampling probability of the ith transition in the replay buffer, ψ_i and ψ_k is the magnitude of the temporal difference error of the ith and kth transitions in the replay buffer [13]. We summarize our algorithm in Algorithm 1.

V. Implementation Details

We compare our algorithms with KLPER, PER, and Vanilla ER. We run ten trials for each algorithm, and to make a fair comparison, we use the same seeds for each algorithm. We implement PER and KLPER algorithms by following the original implementations. We use candidate batch numbers as 4 rather than 8 for KLPER to reduce the computational complexity [16]. We use proportional variant of the Prioritized Experience Replay to evaluate the performance of our algorithm [13]. We note that our algorithm, DPER, does not include the beta parameter that serves for weighted importance sampling [25].

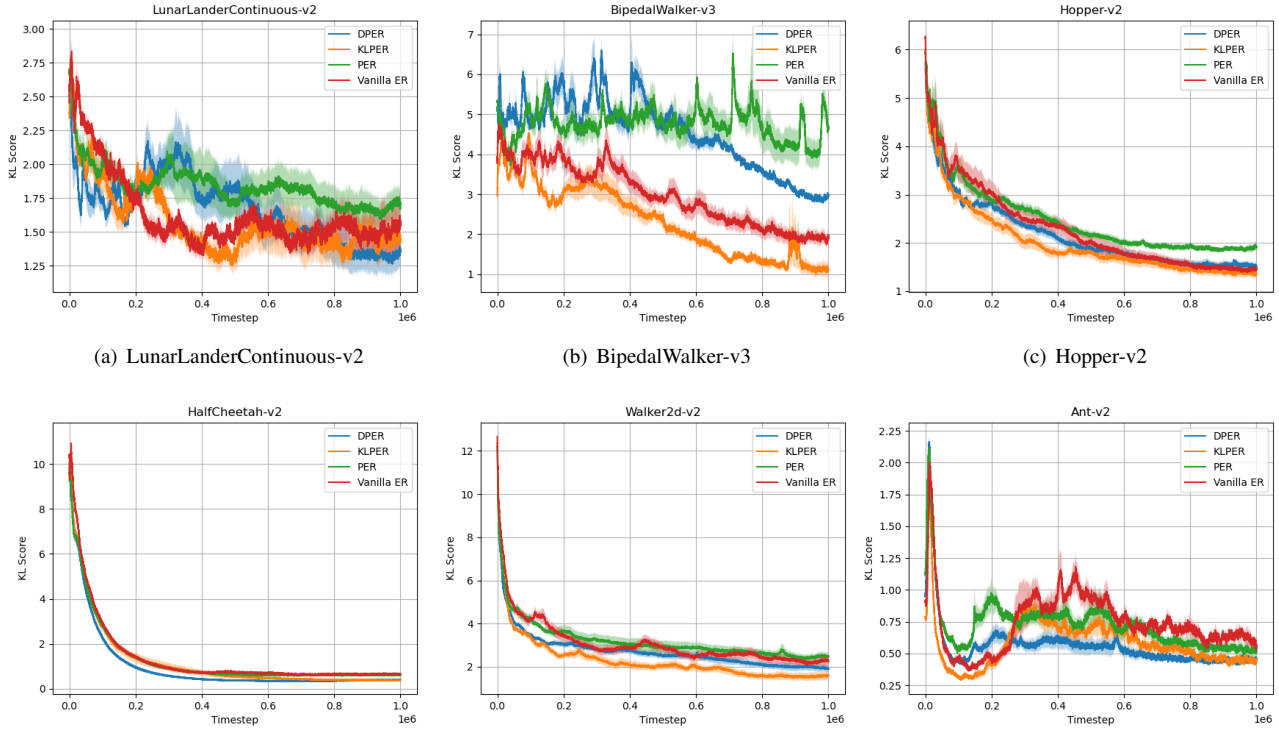
We couple DPER and the other experience replay mechanisms that we use to evaluate DPER with TD3. We also use its original implementation, which is given in the paper that proposes the algorithm [17]. The TD3 has three-layered neural networks for both the Actor and the Critic networks, including two hidden layers with 256 neurons. We use the Adam optimizer [26] for both of the Actor and the Critic. When there are not enough exploration steps, the TD3 algorithm converges to a local optimum policy. As a result, for each experience replay technique, we load the experience memory with 25,000 experiences that the agent collected while exploring the learning environment arbitrarily.



(a) LunarLanderContinuous-v2 (b) BipedalWalker-v3 (c) Hopper-v2

(d) HalfCheetah-v2 (e) Walker2d-v2 (f) Ant-v2

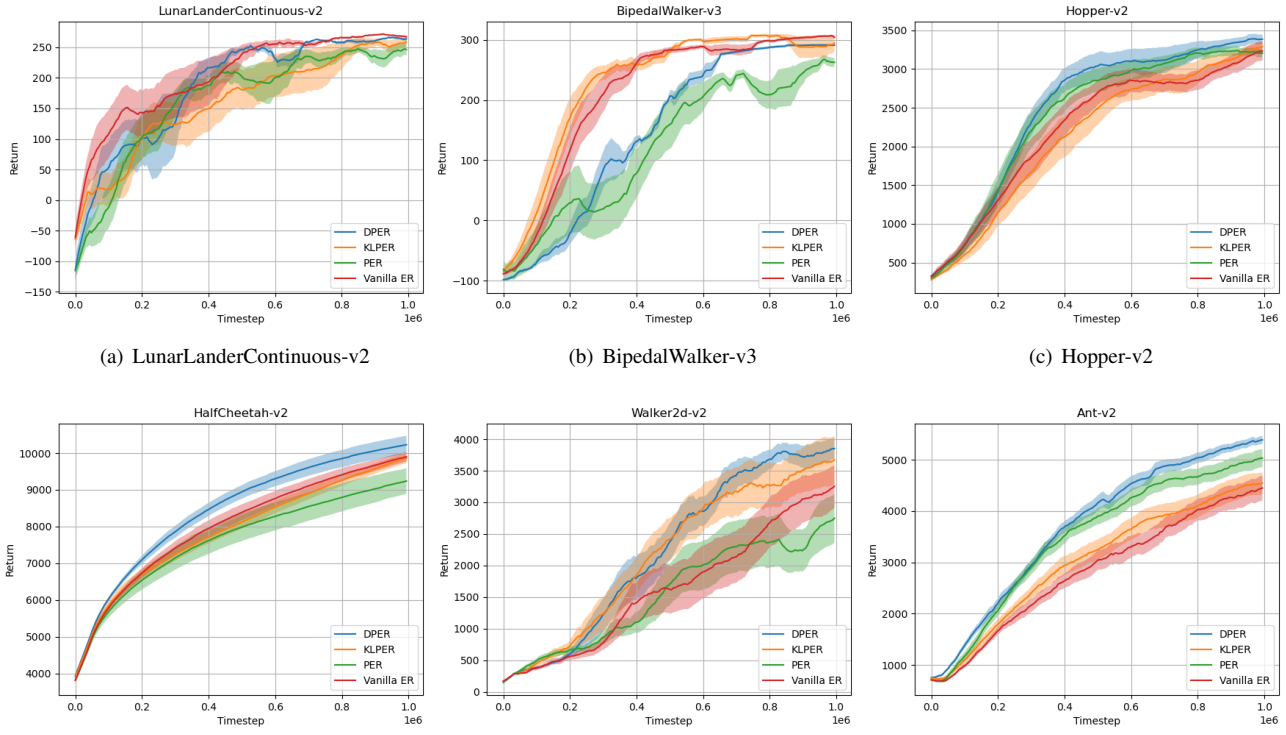
Fig. 3. The average magnitude of the temporal difference error that batch of transitions yield each timestep for DPER and the other experience replay mechanisms that we use to evaluate DPER on six different tasks. The TD3 is the algorithm that is coupled with the experience replay methods. Half a standard deviation over ten trials is represented by the shaded areas.



(a) LunarLanderContinuous-v2 (b) BipedalWalker-v3 (c) Hopper-v2

(d) HalfCheetah-v2 (e) Walker2d-v2 (f) Ant-v2

Fig. 4. η values that batch of transitions yield each timestep for DPER and the other experience replay mechanisms that we use to evaluate DPER on six different tasks. The TD3 is the algorithm that is coupled with the experience replay methods. Half a standard deviation over ten trials is represented by the shaded areas.



(a) LunarLanderContinuous-v2 (b) BipedalWalker-v3 (c) Hopper-v2

(d) HalfCheetah-v2 (e) Walker2d-v2 (f) Ant-v2

Fig. 5. Return curves for DPER and the other experience replay mechanisms that we use to evaluate DPER on six different tasks. The TD3 is the algorithm that is coupled with the experience replay methods. Half a standard deviation over ten trials is represented by the shaded areas.

We select N, which represents the number of candidate batches in our algorithm that is inherited from KLPER, as 4 in our implementation. Since TD3 employs target policy smoothing regularization by adding a noise term when calculating a target value for Critic update, we select the $sigma$ parameter as 0.2.

We execute DPER and the other experience replay mechanisms on six different OpenAI gym and MuJoCo continuous control tasks. We choose the tasks that vary according to their state and action spaces. Every 5,000 timesteps, we assess how well the algorithms work in terms of cumulative reward. Agents perform by applying their current policy ten times during the evaluation episodes. We define an agent's return as the average cumulative reward over ten evaluation episodes.

VI. Experimental Results

In this section, we discuss the metrics we collect during the training, the average magnitude of the temporal difference error and KL scores, and represent learning curves of DPER and the other experience replay mechanisms that we use to evaluate DPER when they combined with the TD3.

We illustrate the average magnitude of the temporal difference error that batch of transitions used for the Critic training at each timestep in Fig. 3. It is apparent that our algorithm and PER choose transitions with higher temporal difference error for every learning task.

We depict the KL divergence value yielded by the batch of the transitions that are used for the Actor training for each timestep in Fig. 4. For all learning tasks, the Actor starts to learn through more off-policy transitions caused by the 25,000 exploration steps. It is confounding that even though our algorithm forces the agent to use more on-policy transitions unambiguously, this is not valid mostly in the BipedalWalker-v3 task. On the other hand, our algorithm is effective in terms of reducing the off-policiness of the learning algorithm for the Ant-v2 and HalfCheetah-v2 tasks.

We emphasize that reducing off-policiness of an algorithm should be inspected carefully. If one component of the experience replay method causes significant changes on-policy of the agent during the training, it may not be possible to reach the desired on-policiness level. This is the case that we encounter for the BipedalWalker-v3 task. DPER is significantly outperformed by the KLPER and Vanilla ER, which both use more on-policy transitions in the BipedalWalker-v3 task. We highlight that our agent barely accomplished using more on-policy batches of transitions and transitions that output higher magnitudes of temporal difference error in Ant-v2 and HalfCheetah-v2 tasks. Moreover, our algorithm significantly outperforms other algorithms in these two environments.

Overall, In four out of six learning tasks, the proposed algorithm, DPER, performs better than the agents employing KLPER, PER, and Vanilla ER. In LunarLanderContinuous-v2 and BipedalWalker-v3 tasks, our algorithm is outperformed by Vanilla ER and KLPER. In these environments, we observe dominant and unexpected policy changes when DPER is coupled with the TD3. The spikes in Fig. 4

represent these changes. Also, PER significantly suffers from the same problem and is outperformed by all of the other algorithms. Therefore, we state that significant policy changes may affect the performances of the agents negatively for all of the experience replay mechanisms we use in this work and should be inspected carefully. However, we leave it as future work.

VII. Conclusion

We propose a method that addresses the drawbacks of other conventional experience replay mechanisms. We improve two mutually exclusive experience replay methods by using them on the Actor and the Critic training separately. We introduce DPER, which decouples the training principles of the Actor and the Critic. Results show that DPER presents promising improvements and outperforms KLPER, PER, and Vanilla ER on particular continuous control tasks. Also, we state that the average magnitude of the temporal difference error and KL scores yielded by the batch of transitions that are used during the training have crucial importance for the learning process.

ACKNOWLEDGMENT

Dogan C. Cicek was supported by Türk Telekom within the 5G and Beyond Graduate Support Programme.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] A. Ghadirzadeh, X. Chen, W. Yin, Z. Yi, M. Björkman, and D. Kragic, "Human-centered collaborative robots with deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 566–571, 2021.

[3] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu, "High-speed autonomous drifting with deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1247–1254, 2020.

[4] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[5] L. Guzman, V. Morellas, and N. Papanikolopoulos, "Robotic embodiment of human-like motor skills via reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3711–3717, 2022.

[6] M. Yuan, J. Shan, and K. Mi, "Deep reinforcement learning based game-theoretic decision-making for autonomous vehicles," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 818–825, 2022.

[7] M. Turan, Y. Almalioglu, H. B. Gilbert, F. Mahmood, N. J. Durr, H. Araujo, A. E. Sarı, A. Ajay, and M. Sitti, "Learning to navigate endoscopic capsule robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3075–3082, 2019.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, p. 293–321, May 1992.

[10] H. Cha, J. Park, H. Kim, M. Bennis, and S.-L. Kim, "Proxy experience replay: Federated distillation for distributed reinforcement learning," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 94–101, 2020.

[11] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning*, pp. 2052–2062, PMLR, 2019.

[12] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[14] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Experience replay optimization," *arXiv preprint arXiv:1906.08387*, 2019.

[15] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.

[16] D. C. Cicek, E. Duran, B. Saglam, F. B. Mutlu, and S. S. Kozat, "Off-policy correction for deep deterministic policy gradient algorithms via batch prioritized experience replay," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1255–1262, 2021.

[17] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

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

[19] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[21] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[22] S.-H. Kong, I. M. A. Nahrendra, and D.-H. Paek, "Enhanced off-policy reinforcement learning with focused experience replay," *IEEE Access*, vol. 9, pp. 93152–93164, 2021.

[23] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[24] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to sample with local and global contexts in experience replay buffer," *arXiv preprint arXiv:2007.07358*, 2020.

[25] A. R. Mahmood, H. P. Van Hasselt, and R. S. Sutton, "Weighted importance sampling for off-policy learning with linear function approximation," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.