Experience Replay Optimization

Daochen Zha, **Kwei-Herng Lai**, **Kaixiong Zhou** and **Xia Hu**
Department of Computer Science and Engineering, Texas A&M University
{daochen.zha, khlai037, zkxiong, xiahu}@tamu.edu

Abstract

Experience replay enables reinforcement learning agents to memorize and reuse past experiences, just as humans replay memories for the situation at hand. Contemporary off-policy algorithms either replay past experiences uniformly or utilize a rule-based replay strategy, which may be sub-optimal. In this work, we consider *learning* a replay policy to optimize the cumulative reward. Replay learning is challenging because the replay memory is noisy and large, and the cumulative reward is unstable. To address these issues, we propose a novel experience replay optimization (ERO) framework which alternately updates two policies: the agent policy, and the replay policy. The agent is updated to maximize the cumulative reward based on the replayed data, while the replay policy is updated to provide the agent with the most useful experiences. The conducted experiments on various continuous control tasks demonstrate the effectiveness of ERO, empirically showing promise in experience replay learning to improve the performance of off-policy reinforcement learning algorithms.

1 Introduction

Experience replay mechanism [Lin, 1992; Lin, 1993] plays a significant role in deep reinforcement learning, enabling the agent to memorize and reuse past experiences. It is demonstrated that experience replay greatly stabilizes the training process and improves the sample efficiency by breaking the temporal correlations [Mnih *et al.*, 2013; Mnih *et al.*, 2015; Wang *et al.*, 2017; Van Hasselt *et al.*, 2016; Andrychowicz *et al.*, 2017; Pan *et al.*, 2018; Sutton and Barto, 2018]. Current off-policy algorithms usually adopt a uniform sampling strategy which replays past experiences with equal frequency. However, the uniform sampling cannot reflect the different importance of past experiences: the agent can usually learn more efficiently from some experiences than from others, just as humans tend to replay crucial experiences and generalize them to the situation at hand [Shohamy and Daw, 2015].

Recently, some rule-based replay strategies have been studied to prioritize important experiences. One approach is to directly prioritize the transitions[1] with higher temporal difference (TD) errors [Schaul *et al.*, 2016]. This simple replay rule is shown to improve the performance of deep Q-network on Atari environments and is demonstrated to be a useful ingredient in Rainbow [Hessel *et al.*, 2018]. Other studies indirectly prioritize experiences through managing the replay memory. Adopting different sizes of the replay memory [Zhang and Sutton, 2017; Liu and Zou, 2017] and selectively remembering/forgetting some experiences with simple rules [Novati and Koumoutsakos, 2018] are shown to affect the performance greatly. However, the rule-based replay strategies may be sub-optimal and may not be able to adapt to different tasks or reinforcement learning algorithms. We are thus interested in studying how we can optimize the replay strategy towards more efficient use of the replay memory.

In the neuroscience domain, the recently proposed normative theory for memory access suggests that a rational agent ought to replay the memories that lead to the most rewarding future decisions [Mattar and Daw, 2018]. For instance, when a human being learns to run, she tends to utilize the memories that can mostly accelerate the learning process; in this context, the memories could relate to walking. We are thus motivated to use the feedback from the environment as a rewarding signal to adjust the replay strategy. Specifically, apart from the agent policy, we consider learning an additional replay policy, aiming at sampling the most useful experiences for the agent to maximize the cumulative reward. A learning-based replay policy is promising because it can potentially find more useful experiences to train the agent and may better adapt to different tasks and algorithms.

However, it is nontrivial to model the replay policy for several challenges. First, transitions in the memory are quite noisy due to the randomness of the environment. Second, the replay memory is typically large. For example, the common memory size for the benchmark off-policy algorithm deep deterministic policy gradient (DDPG) [Lillicrap *et al.*, 2016] can reach 10^6. The replay policy needs to properly filter out the most useful ones among all the transitions in the memory. Third, the cumulative reward is unstable, also due to the environmental randomness. Therefore, it is challenging to effectively and efficiently learn the replay policy.

[1]Transition and experience are considered exchangeable in this work when the context has no ambiguity.

To address the above issues, in this paper, we propose experience replay optimization (ERO) framework, which alternately updates two policies: the agent policy, and the replay policy. Specifically, we investigate how to efficiently replay the most useful experiences from the replay buffer, and how we can make use of the environmental feedback to update the replay policy. The main contributions of this work are summarized as follows:

- Formulate experience replay as a learning problem.
- Propose ERO, a general framework for effective and efficient use of replay memory. A priority vector is maintained to sample subsets of transitions for efficient replaying. The replay policy is updated based on the improvement of cumulative reward.
- Develop an instance of ERO by applying it to the benchmark off-policy algorithm DDPG.
- Conduct experiments on 8 continuous control tasks from OpenAI Gym to evaluate our framework. Empirical results show promise in experience replay learning to improve the performance of off-policy algorithms.

2 Problem Statement

We consider standard reinforcement learning (RL) which is represented by a sextuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_T, \mathcal{R}, \gamma, p_0)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}_T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the state transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, $\gamma \in (0, 1)$ is the discount factor, and p_0 is the distribution of the initial state. At each timestep t, an agent takes action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ and observes the next state s_{t+1} with a scalar reward r_t, which results in a quadruple (s_t, a_t, r_t, s_{t+1}), also called *transition*. This also leads to a *trajectory* of states, actions and rewards $(s_1, a_1, r_1, s_2, a_2, r_2, ...)$. The objective is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the cumulative reward $R = \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t r_t]$. An off-policy agent makes use of an experience replay buffer, denoted as $\mathcal{B}$. At each timestep t, the agent interacts with the environment and stores transition (s_t, a_t, r_t, s_{t+1}) into $\mathcal{B}$. Let $\mathcal{B}_i$ denote the transition in $\mathcal{B}$ at position i. Then for each training step, the agent is updated by using a batch of transitions $\{\mathcal{B}_i\}$ sampled from the buffer.

Based on the notations defined above, we formulate the problem of learning a replay policy as follows. Given a task $\mathcal{T}$, an off-policy agent Λ and the experience replay buffer $\mathcal{B}$, we aim at learning a replay policy ϕ which at each training step samples a batch of transitions $\{\mathcal{B}_i\}$ from $\mathcal{B}$ to train agent Λ, i.e., learning the mapping $\phi : \mathcal{B} \rightarrow \{\mathcal{B}_i\}$, such that better performance can be achieved by Λ on task $\mathcal{T}$ in terms of cumulative reward and efficiency.

3 Methodology

Figure 1 shows an overview of the ERO framework. The core idea of the proposed framework is to use a replay policy to sample a subset of transitions from the buffer, for updating the reinforcement learning agent. Here, the replay policy generates a 0-1 boolean vector to guide subset sampling, where 1 indicates that the corresponding transition is selected (detailed in Section 3.1). The replay policy indirectly teaches the agent by controlling what historical transitions are used, and is adjusted according to the feedback which is defined as the improvement of the performance (detailed in Section 3.2). Our framework is general and can be applied to contemporary off-policy algorithms. Next, we elaborate the details of the proposed ERO framework.

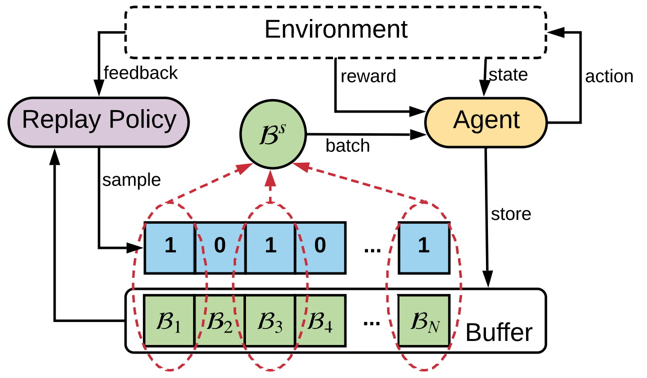

Figure 1: An overview of experience replay optimization (ERO). The reinforcement learning agent interacts with the environment and stores the transition into the buffer. In training, the replay policy generates a vector to sample a subset of transitions $\mathcal{B}^s$ from the buffer, where 1 indicates that the corresponding transition is selected. The sampled transitions are then used to update the agent.

3.1 Sampling with Replay Policy

In this subsection, we formulate a learning-based replay policy to sample transitions from the buffer.

Let $\mathcal{B}_i$ be a transition in buffer $\mathcal{B}$ with a associated feature vector $\mathbf{f}_{\mathcal{B}_i}$ (we empirically select some features, detailed in Section 5), where i denotes the index. In ERO, the replay policy is described as a priority score function $\phi(\mathbf{f}_{\mathcal{B}_i}|\theta^{\phi}) \in (0, 1)$, in which higher value indicates higher probability of a transition being replayed, ϕ denotes a function approximator which is a deep neural network, and θ^{ϕ} are the corresponding parameters. For each $\mathcal{B}_i$, a score is calculated by the replay policy ϕ. We further maintain a vector $\boldsymbol{\lambda}$ to store these scores:

$$\boldsymbol{\lambda} = \{\phi(\mathbf{f}_{\mathcal{B}_i}|\theta^{\phi})|\mathcal{B}_i \in \mathcal{B}\} \in \mathbb{R}^N, \tag{1}$$

where N is the number of transitions in $\mathcal{B}$, element $\boldsymbol{\lambda}_i$ is the priority score of the corresponding transition $\mathcal{B}_i$. Note that it is infeasible to keep all the priority scores up-to-date because the buffer size is usually large. To avoid expensive sweeps over the entire buffer, we adopt an updating strategy similar to [Schaul *et al.*, 2016]. Specifically, a score is updated only when the corresponding transition is replayed. With this approximation, some transitions with very low scores could remain almost never sampled. However, we have to consider the efficiency issue when developing replay policy. In our preliminary experiments, we find this approximation works well and significantly accelerates the sampling. Given the scores vector $\boldsymbol{\lambda}$, we then sample $\mathcal{B}^s$ according to

$$\begin{aligned} &\mathbf{I} \sim \text{Bernoulli}(\boldsymbol{\lambda}), \\ &\mathcal{B}^s = \{\mathcal{B}_i|\mathcal{B}_i \in \mathcal{B} \wedge \mathbf{I}_i = 1\}, \end{aligned} \tag{2}$$

where Bernoulli$(\cdot)$ denotes the Bernoulli distribution, $\mathbf{I}$ is an N-dimensional vector with element $\mathbf{I}_i \in \{0, 1\}$. That is, $\mathcal{B}^s$

Algorithm 1 ERO enhanced DDPG

```
 1: Initialize policy $\pi$, replay policy $\phi$, buffer $\mathcal{B}$
 2: for each iteration do
 3:     for each timestep $t$ do
 4:         Select action $a_t$ according to $\pi$ and state $s_t$
 5:         Execute action $a_t$ and observe $s_{t+1}$ and $r_t$
 6:         Store transition $(s_t, a_t, r_t, s_{t+1})$ into $\mathcal{B}$
 7:         if episode is done then
 8:             Calculate the cumulative reward $r^c_\pi$
 9:             if $r^c_{\pi'} \neq$ null then
10:                 $\mathcal{B}^s$ = UpdateReplayPolicy($r^c_\pi$, $r^c_{\pi'}$, $\mathcal{B}$)
11:             end if
12:             Set $r^c_{\pi'} \leftarrow r^c_\pi$
13:         end if
14:     end for
15:     for each training step do
16:         Uniformly sample a batch $\{\mathcal{B}^s_i\}$ from $\mathcal{B}^s$
17:         Update the critic of $\pi$ with Eq. (9) and (10)
18:         Update the actor of $\pi$ with Eq. (11)
19:         Update target networks with Eq. (12) and (13)
20:         Update $\lambda$ for each transition in $\{\mathcal{B}_i\}$
21:     end for
22: end for
```

is the subset of $\mathcal{B}$ such that the transition is selected when the corresponding value in **I** is 1. In this sense, if transition $\mathcal{B}_i$ has higher priority, it will be more likely to have $\mathbf{I}_i = 1$, hence being more likely to be in the subset $\mathcal{B}^s$. Then $\mathcal{B}^s$ is used to update the agent with standard procedures, i.e., mini-batch updates with uniform sampling. The binary vector **I**, which serves as a "mask" to narrow down all the transitions to a smaller subset of transitions, indirectly affects the relaying.

3.2 Training with Policy Gradient

This subsection describes how the replay policy is updated in ERO framework.

From the perspective of reinforcement learning, the binary vector **I** can be regarded as the "action" taken by the replay policy. The replay-reward is defined as

$$r^r = r^c_\pi - r^c_{\pi'}, \tag{3}$$

where $r^c_{\pi'}$ and r^c_π are the cumulative reward of the previous agent policy π' and the agent policy π updated based on **I**, respectively. The cumulative reward of π is estimated by the recent episodes it performs[2]. The replay-reward can be interpreted as how much the action **I** helps the learning of the agent. The objective of the replay policy is to maximize the improvement:

$$\mathcal{J} = \mathbb{E}_{\mathbf{I}}[r^r]. \tag{4}$$

By using the REINFORCE trick [Williams, 1992], we can calculate the gradient of $\mathcal{J}$ w.r.t θ^ϕ:

$$\begin{aligned}\nabla_{\theta^\phi}\mathcal{J} &= \nabla_{\theta^\phi}\mathbb{E}_{\mathbf{I}}[r^r] \\ &= \mathbb{E}_{\mathbf{I}}[r^r \,\nabla_{\theta^\phi} \log \mathcal{P}(\mathbf{I}|\phi)],\end{aligned} \tag{5}$$

[2]In our implementation, a replay-reward is computed and used to update the replay policy when one episode is finished. r^c_π is estimated by the mean of recent 100 episodes.

Algorithm 2 UpdateReplayPolicy

```
Input:
  Cumulative reward of current policy $r^c_\pi$
  Cumulative reward of previous policy $r^c_{\pi'}$
  Buffer $\mathcal{B}$
Output:
  Sampled subset $\mathcal{B}^s$
1: Calculate replay-reward based on Eq. (3)
2: for each replay updating step do
3:     Randomly sample a batch $\{\mathcal{B}_i\}$ from $\mathcal{B}$
4:     Update replay policy based on Eq. (8)
5: end for
6: Sample a subset $\mathcal{B}^s$ from $\mathcal{B}$ using Eq. (2)
```

where $\mathcal{P}(\mathbf{I}|\phi)$ is the probability of generating binary vector **I** given ϕ, and ϕ is the abbreviation for $\phi(\mathbf{f}_{\mathcal{B}_i}|\theta^\phi)$. Since each entry of **I** is sampled independently based on $\boldsymbol{\lambda}$, the term $\log \mathcal{P}(\mathbf{I}|\phi)$ can be factorized as:

$$\begin{aligned}\log \mathcal{P}(\mathbf{I}|\phi) &= \sum_{i=1}^{N} \log \mathcal{P}(\mathbf{I}_i|\phi) \\ &= \sum_{i=1}^{N} [\mathbf{I}_i \log \phi + (1 - \mathbf{I}_i) \log(1 - \phi)],\end{aligned} \tag{6}$$

where N is the number of the transitions in $\mathcal{B}$, and $\mathbf{I}_i$ is the binary selector for transition $\mathcal{B}_i$. The resulting policy gradient can be written as:

$$\nabla_{\theta^\phi}\mathcal{J} \approx r^r \sum_{i=1}^{N} \nabla_{\theta^\phi}[\mathbf{I}_i \log \phi + (1 - \mathbf{I}_i) \log(1 - \phi)]. \tag{7}$$

If we regard **I** as "labels", then Eq. (7) can be viewed as cross-entropy loss of the replay policy ϕ, scaled by r^r. Intuitively, a current positive (negative) reward will encourage (discourage) the next action to be similar to the current action **I**. Thus, the replay policy is updated to maximize the replay-reward. At each replay updating step, the gradient of Eq. (7) can be approximated by sub-sample a mini-batch of transitions from $\mathcal{B}$ to efficiently update the replay policy:

$$\sum_{j:\mathcal{B}_j \in \mathcal{B}^{batch}} r^r \,\nabla_{\theta^\phi} \,[\mathbf{I}_j \log \phi + (1 - \mathbf{I}_j) \log(1 - \phi)], \tag{8}$$

where $\mathcal{B}^{batch}$ denotes a mini-batch of transitions sampled from $\mathcal{B}$. Note that the replay policy is updated only at the end of each episode. When an episode is finished, the replay-reward for the current action **I** is used to update the replay policy. The updates of the replay policy rely on the cumulative rewards of the recent episodes in the training process, but do not require generating new episodes.

4 Application to Off-policy Algorithms

In this section, we use the benchmark off-policy algorithm DDPG [Lillicrap *et al.*, 2016] as an example to show how ERO is applied. Note that ERO could also be applied to other off-policy algorithms by using similar procedures.

DDPG is a model-free actor-critic algorithm consisting of an actor function $\mu(s|\theta^\mu)$ that specifies the current policy and a critic function $Q(s_t, a_t|\theta^Q)$. Here, θ^μ and θ^Q are approximated by deep neural networks. In training, DDPG optimizes θ^Q by minimizing the following loss w.r.t θ^Q:

$$\mathcal{L}(\theta^Q) = \frac{1}{N}\sum_t (y_t - Q(s_t, a_t|\theta^Q))^2, \tag{9}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^\mu)|\theta^Q). \tag{10}$$

The actor network can be updated by applying chain rules to $\mathcal{J}$, the expected return over the state distribution, w.r.t θ^μ [Silver *et al.*, 2014]:

$$\begin{aligned} \bigtriangledown_{\theta^\mu}\mathcal{J} &\approx \mathbb{E}_{s_t}[\bigtriangledown_{\theta^\mu} Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q)] \\ &= \mathbb{E}_{s_t}[\bigtriangledown_\mu Q(s_t, \mu(s_t|\theta_\mu)|\theta^Q) \bigtriangledown_{\theta_\mu} \mu(s_t|\theta^\mu)]. \end{aligned} \tag{11}$$

To learn the non-linear function approximators in a stable manner, DDPG uses two corresponding target networks to update slowly:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \tag{12}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}, \tag{13}$$

where $\tau \in (0, 1]$, Q' and μ' are the target critic network and target actor network, respectively. DDPG maintains an experience replay buffer. For each training step, the critic and the actor are updated by using a randomly sampled batch of transitions from the buffer.

To apply ERO framework to DDPG, we first sample a subset of the transitions from the buffer according to the replay policy, and then feed this subset of transitions to DDPG algorithm. The ERO enhanced DDPG is summarized in Algorithm 1, where Line 20 updates the transition priority scores with the replay policy, and Algorithm 2 serves for replay learning purpose.

5 Experiments

In this section, we conduct experiments to evaluate ERO. We mainly focus on the following questions: **Q1:** How effective and efficient is our ERO compared with the uniform replay strategy as well as the rule-based alternatives? **Q2:** What kind of transitions could be beneficial based on the learned replay policy?

5.1 Experimental Setting

Our experiments are conducted on the following continuous control tasks from OpenAI Gym[3]: HalfCheetah-v2, InvertedDoublePendulum-v2, Hopper-v2, InvertedPendulum-v2, HumanoidStandup-v2, Reacher-v2, Humanoid-v2, Pendulum-v0 [Todorov *et al.*, 2012; Brockman *et al.*, 2016]. We apply ERO to the benchmark off-policy algorithm DDPG [Lillicrap *et al.*, 2016] for the evaluation purpose. Our ERO is compared against the following baselines:

- **Vanilla-DDPG:** DDPG with uniform sampling.
- **PER-*prop*:** Proportional prioritized experience replay [Schaul *et al.*, 2016], which prioritizes transitions with high temporal difference errors with rules.
- **PER-*rank*:** Rank-based prioritized experience replay [Schaul *et al.*, 2016], a variant of PER-*prop* which adopts a binary heap for the ranking purpose.

For a fair comparison, PER-*prop*, PER-*rank* and ERO are all implemented on the identical Vanilla-DDPG.

Implementation details Our implementations are based on OpenAI DDPG baseline [4]. For Vanilla-DDPG, we follow all the settings described as in the original work [Lillicrap *et al.*, 2016]. Specifically, $\tau = 0.001$ is used for soft target updates, learning rates of 10^{-4} and 10^{-3} are adopted for actor and critic respectively, the Ornstein-Uhlenbeck noise with $\theta = 0.15$ and $\sigma = 0.2$ is used for exploration, the mini-batch size is 64, the replay buffer size is 10^6, the number of rollout steps is 100, and the number of training steps is 50. For other hyperparameters and the network architecture, we use the default setting as in the OpenAI baseline. For the two prioritized experience replay methods PER-*prop* and PER-*rank*, we search the combinations of α and β and report the best results. For our ERO, we empirically use three features for each transition: the reward of the transition, the temporal difference (TD) error, and the current timestep. TD error is updated only when a transition is replayed. We implement the replay policy by using MLP with two hidden layers (64-64). The number of replay updating steps is set to 1 with minibatch size 64. Adam optimizer is used with a learning rate of 10^{-4}. Our experiments are performed on a server with 24 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.2GHz processors and 4 GeForce GTX-1080 Ti 12 GB GPU.

5.2 Performance Comparison

The learning curves on 8 classical control tasks are shown in Figure 2. Each task is run for 5 times with 2×10^6 timesteps using different random seeds, and the average return over episodes is reported. We make the following observations.

First, the proposed ERO consistently outperforms all the baselines on most of the continuous control tasks in terms of sample efficiency. On HalfCheetah, InvertedPendulum, and InvertedDoublePendulum, ERO performs clearly better than Vanilla-DDPG. On Hopper, Pendulum, and Humanmoid, faster learning speed with a higher return is observed. The intuition of our superior performance on various environments is that the replay policy of ERO is updated to replay the most suitable transitions during training. This also demonstrates that exploiting the improvement of the policy as a reward to update the replay policy is promising.

Second, rule-based prioritized replay strategies do not provide clear benefits to DDPG on the 8 continuous control tasks, which can also be verified from the results in [Novati and Koumoutsakos, 2018]. Specifically, PER-*prop* only provides a very slight improvement on 5 out of 8 tasks, and shows no clear improvement on the others. An interesting observation is that PER-*rank* even worsens the performance on 4 out

[3]https://gym.openai.com/

[4]https://github.com/openai/baselines

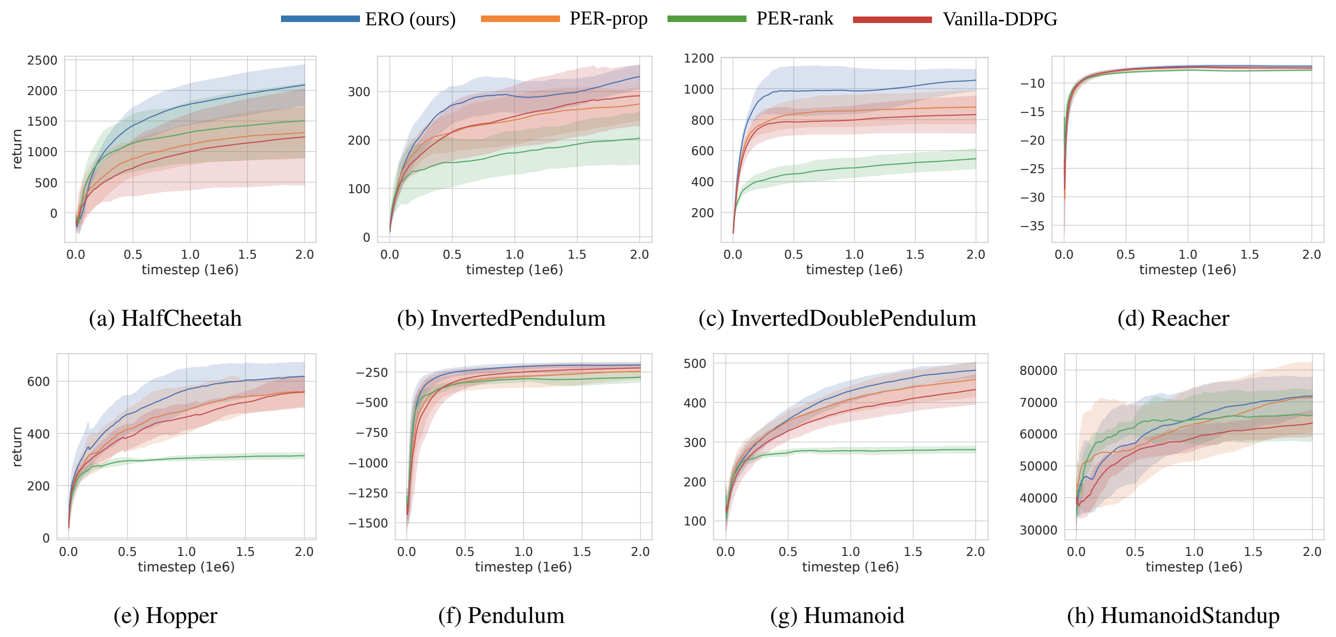

Figure 2: Performance comparison of ERO against baselines on 8 continuous control tasks. The shaded area represents mean $\pm$ standard deviation. ERO outperforms baselines on most of the continuous control tasks.

of 8 continuous control tasks. A possible explanation is that the rank-based prioritization samples transitions based on a power-law distribution. When the buffer size is large, the transitions near the tail are barely selected, which may lead to significant bias in the learning process[5]. Recall that the two rule-based strategies prioritize the transitions with high temporal difference (TD) errors. A potential concern is that a transition with high TD error may substantially deviate from the current policy, hence introducing noise to the training, especially in the later training stages when the agent focuses on a preferred state space. This may partly explain why the two PER methods do not show significant improvement over Vanilla-DDPG. The results further suggest that rule-based replay strategies may be sensitive to different agents or environments. By contrary, the replay policy of ERO is updated based on the feedback from the environment, and consistent improvement is observed in various tasks.

Comparison of running time Table 3 shows the average running time in seconds of ERO enhanced DDPG and Vanilla-DDPG on the 8 continuous control tasks for 2×10^6 timesteps. We observe that ERO requires slightly more running time than Vanilla-DDPG. It is expected because ERO requires additional computation for replay policy update. Note that the interactions with the environment usually dominate the cost of RL algorithms. In practice, the slightly more computational resources of ERO, which are often much cheaper than the interactions with the environment, are greatly outweighed by the sample efficiency it provides.

[5]We have tuned the hyperparameters of the importance sampling trying to correct the potential bias, but we did not observe better performance.

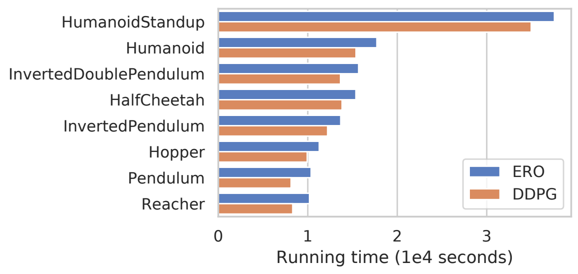

Figure 3: Comparison of running time in seconds for ERO and Vanilla-DDPG. ERO requires only slightly more running time compared with Vanilla-DDPG.

5.3 Analysis of the Sampled Transitions

For further understanding the learned replay policy, we record the transition features through the entire training process of one trial on HalfCheetah to analyze the characteristics of the sampled transitions. Figure 4 plots the average values for the three features: temporal difference (TD) error, timestep difference between the training timestep and the timestep at which the transition is obtained, and reward of the transition. We may empirically find some insights of the replay policy.

As expected, PER-*prop* tends to sample transitions with high TD errors. Interestingly, the learned replay policy of ERO samples more transitions with low TD errors, which are even lower than Vanilla-DDPG. It contradicts to our belief that the transitions with low TD errors are well-known by the agent, which could cause catastrophic forgetting problem. It also contradicts to the central idea of prioritized replay which prioritizes “unexpected” transitions. We hypothesize this task

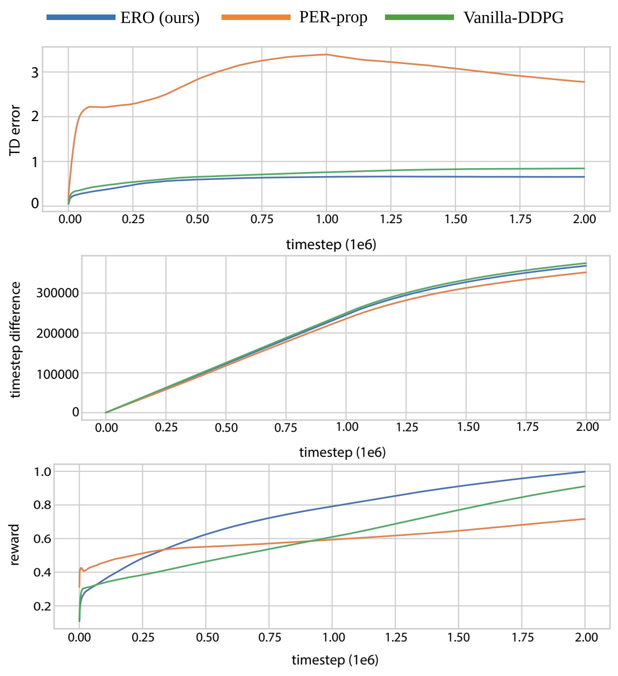

Figure 4: The evolution of TD error (top), timestep difference (middle), and reward of the transition (bottom) with respect to timestep. Timestep difference is the difference between the training timestep and the timestep at which the transition is obtained (that is, low timestep difference means that the agent tends to use recent transitions for the updates). The average values over transitions are plotted for clearer visualization.

may not necessarily contain catastrophic states, and the transitions with slightly low TD errors may align better with the current policy and could be more suitable for training in this specific task. The unsatisfactory performance of PER methods also suggests that prioritizing transions with high TD errors may not help in this task. We believe more studies are needed to understand this aspect in the future work. We observe that both PER-*prop* and ERO sample more recent transitions than Vanilla-DDPG. This is also expected for PER-*prop*, because newly added transitions usually have higher TD errors. Although ERO tends to sample transitions with low TD errors, it also favors recent transitions. This suggests that recent transitions may be more helpful in this specific task. For the transition reward, all the three methods tend to sample transitions with higher rewards as timestep increases. It is reasonable since more transitions with higher rewards are stored into the buffer in the later training stages.

Overall, we find that it is nontrivial to heuristically use rules to define a proper replay strategy, which may depend on a lot of factors from both the algorithm and the environment. A simple rule may not be able to identify the most useful transitions and could be sensitive to different tasks or different reinforcement learning algorithms. For example, the prioritized experience replay shows no clear benefit to DDPG on the above continuous control tasks from OpenAI gym. A learning-based replay policy will be more desirable because it is optimized based on different tasks and algorithms.

6 Discussion and Extension

Replay learning can be regarded as a kind of meta-learning. There are several analogous studies in the context of RL: learning to generate data for exploration [Xu *et al.*, 2018a], learning intrinsic reward to maximize the extrinsic reward on the environment [Zheng *et al.*, 2018], adapting discount factor [Xu *et al.*, 2018b]. Different from these studies, we focus on how reinforcement learning agents can benefit from learning-based replay policy.

Our ERO can also be viewed as a teacher-student framework, where the replay policy (teacher) provides past experiences to the agent (student) for training. Our work has similarity to learning a teaching strategy in the context of supervised learning [Fan *et al.*, 2018; Wu *et al.*, 2018]. However, replay learning differs from teaching a supervised classifier. It is much more challenging due to the large and noisy replay memory, and the unstable learning signal. Our framework could be extended to teach the agent when to remember/forget experiences, or what buffer size to adopt.

ERO could potentially motivate the research of continual (lifelong) learning. Experience replay is shown to effectively transfer knowledge and mitigate forgetting [Yin and Pan, 2017; Isele and Cosgun, 2018]. While existing studies selectively store/replay experiences with rules, it is possible to extend ERO to a learning-based experience replay in the context of continual learning.

7 Conclusion and Future Work

In this paper, we identify the problem of experience replay learning for off-policy algorithms. We introduce a simple and general framework ERO, which efficiently replays the useful experiences to accelerate the learning process. We develop an instance of our framework by applying it to DDPG. Experimental results suggest that ERO consistently improves the sample efficiency compared with Vanilla-DDPG and rule-based strategies. While more studies are needed to understand experience replay, we believe our results empirically show that it is promising to learn experience replay for off-policy algorithms. The direct future work is to extend our framework to learn which transitions should be stored/removed from the buffer and how we can adjust the buffer size in different training stages. We are also interested in studying learning-based experience replay in the context of continual learning. We will investigate the effectiveness of more features, such as signals from the agent, to study which features are important. Finally, since our framework is general, we would like to test it on other off-policy algorithms.

Acknowledgements

This work is, in part, supported by NSF (#IIS-1718840 and #IIS-1750074). The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

References

[Andrychowicz *et al.*, 2017] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter

Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, 2017.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Fan *et al.*, 2018] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. In *ICLR*, 2018.

[Hessel *et al.*, 2018] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[Isele and Cosgun, 2018] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI*, 2018.

[Lillicrap *et al.*, 2016] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[Lin, 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[Lin, 1993] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[Liu and Zou, 2017] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. *arXiv preprint arXiv:1710.06574*, 2017.

[Mattar and Daw, 2018] Marcelo Gomes Mattar and Nathaniel D Daw. Prioritized memory access explains planning and hippocampal replay. *bioRxiv*, page 225664, 2018.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Novati and Koumoutsakos, 2018] Guido Novati and Petros Koumoutsakos. Remember and forget for experience replay. *arXiv preprint arXiv:1807.05827*, 2018.

[Pan *et al.*, 2018] Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. In *IJCAI*, 2018.

[Schaul *et al.*, 2016] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICML*, 2016.

[Shohamy and Daw, 2015] Daphna Shohamy and Nathaniel D Daw. Integrating memories to guide decisions. *Current Opinion in Behavioral Sciences*, 5:85–90, 2015.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[Wang *et al.*, 2017] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *ICLR*, 2017.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[Wu *et al.*, 2018] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Lai Jian-Huang, and Tie-Yan Liu. Learning to teach with dynamic loss functions. In *NeurIPS*, 2018.

[Xu *et al.*, 2018a] Tianbing Xu, Qiang Liu, Liang Zhao, Wei Xu, and Jian Peng. Learning to explore with meta-policy gradient. In *ICML*, 2018.

[Xu *et al.*, 2018b] Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, 2018.

[Yin and Pan, 2017] Haiyan Yin and Sinno Jialin Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *AAAI*, 2017.

[Zhang and Sutton, 2017] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *NIPS Deep Reinforcement Learning Symposium*, 2017.

[Zheng *et al.*, 2018] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*, 2018.