Attentive Experience Replay

Peiquan Sun, Wengang Zhou, Houqiang Li
CAS Key Laboratory of Technology in GIPAS, EEIS Department, University of Science and Technology of China
spq@mail.ustc.edu.cn, {zhwg, lihq}@ustc.edu.cn

Abstract

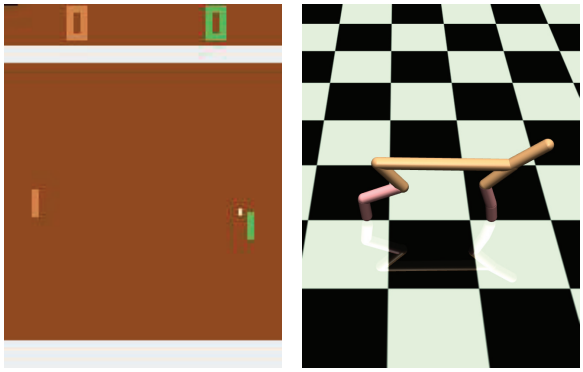
Experience replay (ER) has become an important component of deep reinforcement learning (RL) algorithms. ER enables RL algorithms to reuse past experiences for the update of current policy. By reusing a previous state for training, the RL agent would learn more accurate value estimation and better decision on that state. However, as the policy is continually updated, some states in past experiences become rarely visited, and optimization over these states might not improve the overall performance of current policy. To tackle this issue, we propose a new replay strategy to prioritize the transitions that contain states frequently visited by current policy. We introduce *Attentive Experience Replay* (AER), a novel experience replay algorithm that samples transitions according to the similarities between their states and the agent's state. We couple AER with different off-policy algorithms and demonstrate that AER makes consistent improvements on the suite of OpenAI gym tasks.

1 Introduction

Deep reinforcement learning (RL) has made remarkable advances in many sequential decision-making problems, including Atari games (Mnih et al. 2013; 2015), realistic simulated robotic (Schulman et al. 2015; 2017) and board games (Silver et al. 2016; 2017). With the utilization of deep neural network (DNN) as function approximators, deep RL algorithms are able to learn complex nonlinear policies (or value functions) directly from high dimensional input without prior knowledges. *Experience Replay* (ER) (Lin 1992) is a technique that stores and reuses past experiences with a replay buffer. By randomly sampling transitions from the replay buffer, ER alleviates the temporal correlations between sequential transitions, and offers i.i.d. samples required for the training of DNN. Furthermore, the reuse of the transitions from the past improves the sample efficiency and stabilizes the learning process.

As suggested in (De Bruin et al. 2018), the learning efficiency and stability of RL algorithm, as well as the final performance of the learned policy, are heavily dependent on how the experiences are replayed. In the original form of ER, experience transitions are replayed uniformly at random from the replay buffer, which neglects the significance of different transitions. To make better use of the experiences, various sampling strategies (De Bruin et al. 2015; Schaul et al. 2016; De Bruin et al. 2016; Novati and Koumoutsakos 2019; Zha et al. 2019) have been proposed to prioritize important experiences. For off-policy algorithms, as the policy diverges from previous behaviors, some experiences in the replay buffer might become irrelevant to current policy. For example, some transitions from the past might contain states that would never be visited by current policy. Optimization over these states might not improve the overall performance of current policy and can possibly undermine the performances on the frequently visited states. For off-policy algorithms that contain such irrelevant experiences, a proper sampling strategy should be able to improve the performance.

In this paper, we propose a new criterion for sampling transitions from the replay buffer. We argue that transitions that contain states frequently visited by current policy should be sampled with higher priorities, and ones that contain rarely visited states should be sampled less. Based on



(a) Pong (b) HalfCheetah

Figure 1: Snapshots of OpenAI gym tasks. **Left**: Pong, one of the Atari 2600 games. **Right**: HalfCheetah, one of the MuJoCo tasks.

this criterion, we propose a new experience replay algorithm called *Attentive Experience Replay* (AER). AER computes the similarities between the states in past transitions and the agent's state, and implicitly assigns high priorities to the similar transitions. To evaluate the performance of the proposed algorithm, we couple AER with four off-policy algorithms: DQN, SAC, TD3 and DDPG. We compare AER with two ER methods, the uniform sampling and the *prioritized experience replay* (PER) (Schaul et al. 2016), on the suite of OpenAI gym tasks (Figure 1) (Brockman et al. 2016). The results show that our method achieves better performance and scalability, outperforming the uniform sampling and the PER method over different algorithms and tasks.

2 Related Work

In this section, we briefly review the related methods. First, we present the formulation of the reinforcement learning problem. After that, we introduce the related off-policy RL algorithms and experience replay methods, respectively.

2.1 Reinforcement Learning Formulation

In reinforcement learning, an agent interacts with the environment with the aim of maximizing the cumulative reward. At each discrete time step t, the agent observes its state $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}$ following its policy $a_t \sim \pi(a|s_t)$, receiving a reward $r_t \in \mathbb{R}$ and observing a new state $s_{t+1} \in \mathcal{S}$ according to the environment's dynamic $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$, which results in an experience transition quadruple $e_t = (s_t, a_t, r_t, s_{t+1})$. The return is defined as the discounted cumulative reward:

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

where γ is a discount factor determining the priority of short-term rewards. The expected return after taking an a_t at state s_t following policy π is formulated as follows:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi}[R_t|s_t, a_t]. \tag{2}$$

The corresponding value of the state s_t is defined as follows:

$$V^{\pi}(s_t) = \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi}[R_t|s_t]. \tag{3}$$

The objective of reinforcement learning is to find a policy π that maximizes the expected cumulative reward:

$$J(\pi) = \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi}[R_0] = \sum_{i=0}^{T} \mathbb{E}_{s_i\sim\mathcal{P}, a\sim\pi(a|s_i)}\left[\gamma^i r(s_i, a_i)\right]. \tag{4}$$

2.2 Deep RL Algorithms with Experience Replay

The combination of deep RL algorithms with experience replay has shown great success on different RL tasks. In this work, we focus on four deep off-policy algorithms.

Deep Q-network. In deep Q-network (DQN) algorithm (Mnih et al. 2013; 2015), a deep neural network is used to approximate the optimal value function:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \tag{5}$$

after experiencing a state s and taking an action a. The deep Q-network $Q(s, a; \theta)$ is parametrized using a deep neural network, in which θ are the parameters. During training, at each time step t, the DQN agent stores its experience $e_t = (s_t, a_t, r_t, s_t + 1)$ into a replay buffer $\mathcal{D} = \{e_1, e_2 \cdots\}$ that holds the last one million transitions. When performing updates, mini-batches of experience $(s, a, r, s') \sim U(\mathcal{D})$ are uniformly sampled from the replay buffer to optimize the deep Q-network using stochastic gradient descent by minimizing loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[\left(y - Q(s, a; \theta)\right)^2\right], \tag{6}$$

where $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the bootstrapping target, and θ^- represents the parameters of the *target network* $Q^-(s, a; \theta^-)$, a periodic copy of the deep Q-network $Q(s, a; \theta)$. With the merits of combining deep RL algorithm with experience replay, DQN and its variants (Hessel et al. 2018; Van Hasselt, Guez, and Silver 2016) show superhuman performance on the Atari 2600 games.

Deep deterministic policy gradient. Deep deterministic policy gradient (DDPG) (Lillicrap et al. 2016) is an off-policy actor-critic algorithm that extends the deterministic policy gradient (DPG) algorithm (Silver et al. 2014) by using deep neural networks as function approximators. DDPG contains two neural networks, *i.e.*, the value-network (*a.k.a.* critic) that outputs $Q(s, a; \theta)$, and the policy-network (*a.k.a.* actor) that selects actions based on given states $a = \mu(s; \phi)$. DDPG alternately updates these two networks. The value-network is trained using the loss function similar to DQN:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[\left(y - Q(s, a; \theta)\right)^2\right], \tag{7}$$

where $y = r + \gamma Q(s', \mu(s'; \phi^-); \theta^-)$. $Q(s, a; \theta^-)$ and $\mu(s; \phi^-)$ are the target networks. The policy-network is trained to output an action a that maximizes the value predicted by the value-network:

$$\begin{aligned}\nabla_{\phi} J(\mu) &\approx \nabla_{\phi} \mathbb{E}_{s\sim U(\mathcal{D})}\left[Q(s, a|\theta)\,|_{a=\mu(s|\phi)}\right] \\ &= \nabla_{\phi} \mathbb{E}_{s\sim U(\mathcal{D})}\left[\nabla_a Q(s, a|\theta)_{a=\mu(s)} \nabla_{\phi} \mu(s|\phi)\right].\end{aligned} \tag{8}$$

TD3. Twin delayed deep deterministic policy gradient (TD3) (Fujimoto, van Hoof, and Meger 2018) makes several improvements on DDPG to alleviate the overestimation of the value-network. Motivated by double Q-learning (Hasselt 2010; Van Hasselt, Guez, and Silver 2016) that prevents the overestimation in Q-learning, TD3 introduces *clipped double Q-learning*, which learns two separate value-networks. The update target of the clipped double Q-learning is formulated as:

$$y = r + \gamma \min_{i=1,2} Q\left(s', \mu\left(s'; \phi_i\right); \theta_i\right), \tag{9}$$

which is the minimum estimation among two value-functions.

Soft actor-critic. Soft actor-critic (SAC) (Haarnoja et al. 2018) is an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework. SAC incorporates the policy entropy into the optimization objective:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{s_t \sim \mathcal{P}, a_t \sim \pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot|\mathbf{s}_t\right)\right) \right], \quad (10)$$

where $\mathcal{H}\left(\pi\left(\cdot|\mathbf{s}_t\right)\right)$ is the entropy measure of the policy π and α is the temperature parameter that determines the relative importance of the entropy term against the reward. Optimizing such objective encourages the agent to act as randomly as possible while maximizing the cumulative reward, and hence improves the robustness and stability of the policy.

2.3 Experience Replay Methods

In the original form of ER, transitions are uniformly sampled from the replay buffer. However, intuitively, some transitions are more important than others. An ideal experience replay strategy is to sample transitions according to how much current agent can learn from them. While such measure is not directly accessible, many proxies have been proposed from two perspectives: the first is how to retain experiences in the replay buffer, and the second is how to sample experiences from the buffer. Motivated by different optimization objectives, different replay strategies have been proposed.

In simple continuous control tasks, De Bruin et al. (2015; 2016) suggest that replay buffer should contain variant transitions that are not close to current policy to prevent fitting to local minimums and the best replay distribution is in between on-policy distribution and uniform distribution. However, the authors state that this method is not suitable for complex tasks where policy is updated for many iterations. For lifelong learning, a good replay strategy should retain the transitions from previous tasks and maximize the coverage of the state space (Isele and Cosgun 2018). In RL problems, when the rewards are sparse, agent can learn from failed experiences by reproducing artificial successful trajectories, that is, replacing the original goals with states in the trajectories (Andrychowicz et al. 2017).

For complex control tasks, *prioritized experience replay* (PER) (Schaul et al. 2016) measures the importances of the transitions using the magnitude of temporal-difference (TD) error:

$$|\delta| = \left| r + \gamma Q\left(s', a'; \theta^{-}\right) - Q\left(s, a; \theta\right) \right|. \quad (11)$$

And the probability of sampling transition i is define as:

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \quad (12)$$

where $p_i = |\delta_i| + \epsilon$ is the priority of transition i. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform sampling. *Remember and forget experience replay* (ReF-ER) (Novati and Koumoutsakos 2019) actively enforces the similarity between policy and the transitions in the replay buffer, since on-policy transitions are more useful for the training of current policy. *Experience replay optimization* (ERO) (Zha et al. 2019) can be regarded as a kind of meta-learning. Besides the agent policy, ERO also learns a replay policy. Replay policy is trained to provide the agent with the most useful experiences. In this paper, we compare our proposed method AER with vanilla ER and PER on different off-policy algorithms. For ReF-ER and ERO, we cannot readily couple them with SAC, TD3 and DQN. Hence we leave the comparison with these two methods to our future work.

3 Attentive Experience Replay

3.1 Motivation

In reinforcement learning tasks, the learning agent interacts with the environment following its policy. For different policies, states are visited with different frequencies. For a certain policy π, the induced state distribution is defined as follows:

$$\rho^{\pi}(s) = (1-\gamma) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p_0(s') p\left(s' \to s, t, \pi\right) \mathrm{d}s', \quad (13)$$

where $p_0(s')$ is the distribution of the initial state. In complex control tasks, each policy only explores a subset of the high dimensional state-space. As a result, some states are frequently visited by current policy, while other states rarely appear. To improve the policy, it is desirable to improve the performances on frequently visited states, that is, making more accurate value estimations and better action selections.

Off-policy algorithms utilize deep neural networks (DNNs) as value function approximators, and recall past experiences stored in a replay buffer $\mathcal{D}$ to compute the gradients for the updates of DNNs. DNNs tend to have lower estimation errors on states similar to those on which they have been trained (Burda et al. 2019b). Based on this observation, to make better performances on frequently visited states, we want the frequently visited states to be sampled with higher priorities. However, as the agent continues to update its policy, current policy π is increasingly dissimilar from previous behaviors. As a consequence, the state distribution $\rho^{\mathcal{D}}(s)$ in the replay buffer differs from on-policy distribution $\rho^{\pi}(s)$. Uniformly sampling transitions from the buffer for training might cause the DNN to fit to some undesired states. Some strategies have been proposed to remedy the inconsistency between $\rho^{\mathcal{D}}(s)$ and $\rho^{\pi}(s)$ by slowing down the change of the policy, including meticulous learning rate tunning (Wang et al. 2016), and slowly changing target network (Mnih et al. 2015). ReF-ER (Novati and Koumoutsakos 2019) proposes to enforce the similarity between transitions in the replay and current policy, and select "near-policy" transitions for training.

In this work, we present a new criterion of selecting transitions for the training of current policy, which is assigning transitions with higher priorities, if the transitions contain states frequently visited by current policy. Specifically, for a transition (s, a, r, s'), if $\rho^{\pi}(s)$ is large, which indicates that it is frequently visited by policy π, we will classify it as an "on-distribution" transition and use it for training more often; if $\rho^{\pi}(s) \approx 0$, which indicates that it is rarely visited, this transition will be classified as an "off-distribution" transition and rarely be sampled. Based on this rule, we propose

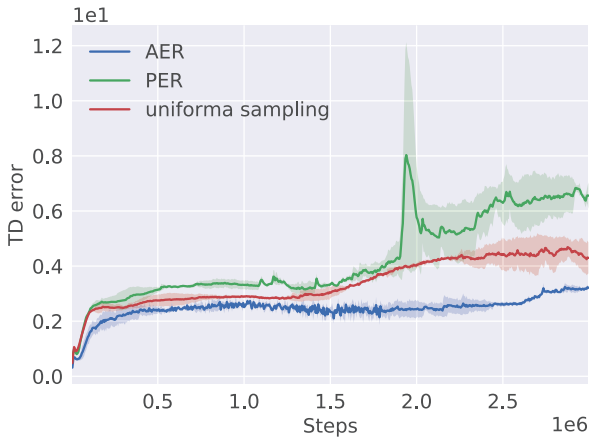


Figure 2: The comparison of TD errors of the sampled transition on *HalfCheetah-v2* task.

Algorithm 1 Attentive experience replay

Input: minibatch k, learning rate η, replay period K and buffer size N, coefficient λ, total steps T, similarity measure $\mathcal{F}$
Initialize replay memory $\mathcal{D} = \emptyset$
Observe s_0 and choose $a_0 \sim \pi(a|s_0; \phi)$
for $t = 1$ **to** T **do**
  Observe s_t, r_t
  Store transition $(s_{t-1}, a_{t-1}, r_t, s_t)$ in $\mathcal{D}$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** $\lambda \cdot k$ **do**
      Uniformly Sample transitions j: $\left(s_j, a_j, r_j, s'_j\right)$
      Compute similarity $l_j = \mathcal{F}(s_j, s_t)$
    end for
    Select k most similar transitions $\mathcal{B}$
    Calculate weight-change Δ using transitions $\mathcal{B}$
    Update weights $\phi \leftarrow \phi + \lambda \cdot \eta \cdot \Delta$
  end if
  Choose action $a_t \sim \pi(a|s_t; \phi)$
end for

Attentive Experience Replay (AER) algorithm. AER adopts a new sampling strategy called *attentive sampling* that samples on-distribution transitions with higher priorities.

3.2 Attentive Sampling

Ideally, if we know the exact state distribution $\rho^\pi(s)$, we can assign priorities according to this distribution. However, this distribution is not directly accessible, especially when the policy is continuously changing. We propose to use the similarity between transition states and current state s_t as a reasonable proxy to sample frequently visited states more often. The reasons are as follows. First, current state s_t is an on-distribution state, since it is currently visited by the policy; and similar states usually have similar densities under distribution $\rho^\pi(s)$. An intuition is that sequential states are similar to each other, and they are all on-distribution states since they have all been visited by current policy. Second, with the generalization ability of DNN, training on similar states also improves the performance on state s_t. Therefore, training on transitions sampled using AER improves the performance on s_t, and in consequence, improves the performance of current policy.

Based on this surrogate measure, we propose an *attentive sampling* strategy. When performing updates, we uniformly sample $\lambda \cdot k$ instead of k transitions from the replay buffer. Then we compute the similarities between on-distribution state s and the sampled transitions, and choose the k most similar transitions for training (see Algorithm 1). Here $\lambda \geq 1$ determines how much prioritization to on-distribution transitions is used, with $\lambda = 1$ corresponding to the uniform sampling, with $\lambda = N$, where N is the replay buffer size, corresponding to sampling the k most similar transitions in the replay buffer.

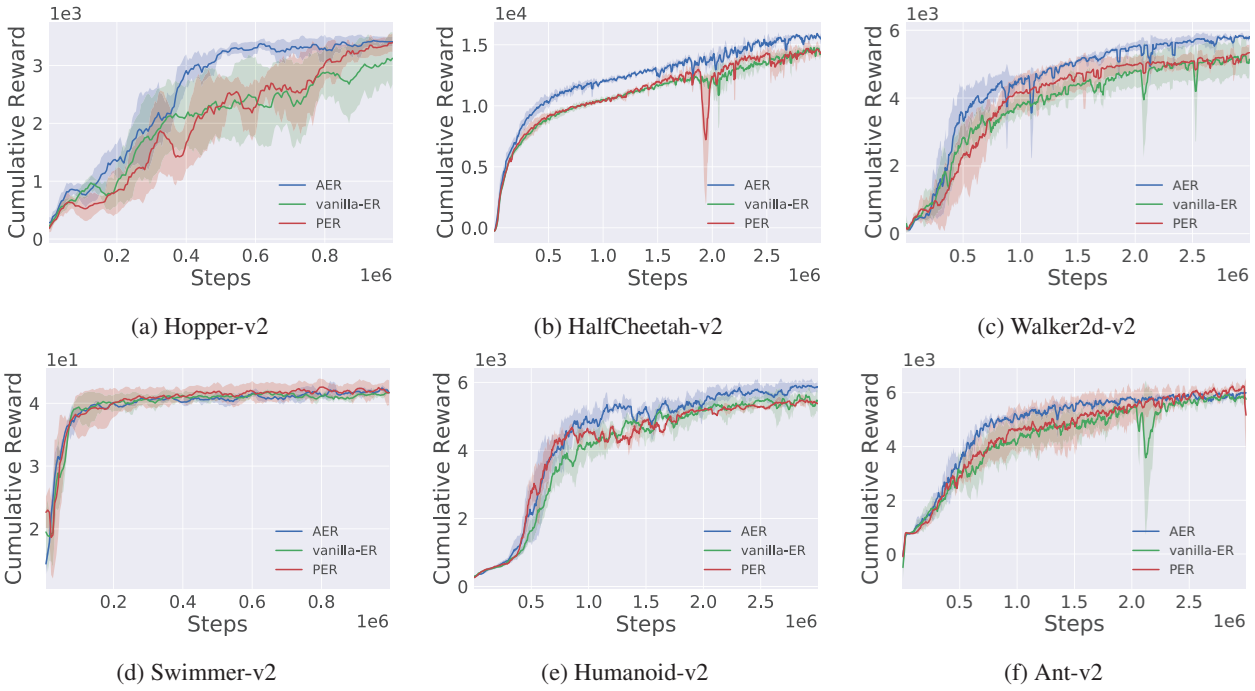
3.3 Bias Annealing

AER deviates from uniform sampling by assigning some transitions with higher priorities. As pointed out in previous work (Schaul et al. 2016), such deviation introduces bias and can change the solution that the policy will converge to. But in fact, in reinforcement learning, the training process is highly non-stationary due to the changing policies, state distributions and update targets. The unbiased nature of updates is most important near convergence of the policy. Hence, the small bias introduced by attentive sampling is negligible when the policy is still changing. To make sure that attentive sample does not introduce bias when the policy converges, we propose to anneal λ over time. In practice, we linearly anneal λ from a initial value λ_0 to 1 in $\alpha \cdot T$ steps. Here T is the total training steps, and $\alpha \leq 1$ is the fraction of total steps to perform attentive sampling. The choice of α depends on how fast the policy converges; a smaller α corresponds to faster policy convergence.

Since AER continuously samples on-distribution transitions and trains the neural network, the neural network should make small prediction error on the sampled transitions, which means that the TD errors should be small. As shown in Figure 2, PER samples transitions with higher TD errors as expected, and transitions sampled by AER have smallest TD errors. Since smaller TD error yields smaller gradient, AER uses a larger learning rate to compensate, while PER uses a smaller learning rate. In practice, we use $\lambda \cdot \eta$ as the learning rate, where η is the learning rate of uniform sampling; since we anneal λ over time, we also decrease the learning rate to its default η. However, annealing the learning rate is a common trick to speed up the training. To isolate the effect of learning rate annealing, we conduct experiments (see Section 4.4 for details) to show that annealing the learning rate itself does not improve the learning efficiency.

4 Experiments

In this section, we couple AER, PER and vanilla-ER (uniform sampling) with four off-policy deep RL algorithms: SAC, TD3, DDPG and DQN. We conducted several experiments to evaluate their performances and aimed to show



(a) Hopper-v2 (b) HalfCheetah-v2 (c) Walker2d-v2

(d) Swimmer-v2 (e) Humanoid-v2 (f) Ant-v2

Figure 3: Learning curves for SAC with AER, vanilla-ER and PER on MuJoCo tasks. The solid lines indicate the mean across 5 random seeds and the shaded areas show the standard error.

how different experience replay methods can affect the performances of different off-policy algorithms. For deep actor-critic algorithms, we tested on the MuJoCo (Todorov, Erez, and Tassa 2012) tasks implemented in OpenAI Gym (Brockman et al. 2016), and for DQN, we measured the performances on the Atari 2600 games. The results are demonstrated by plotting the mean cumulative reward. For the plots, the solid lines indicate the average cumulative rewards among the training process averaged over 5 different random seeded training trials and the shaded areas show the standard error. All experiments are performed on a server with 40 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.4GHz processors and 8 GeForce GTX-1080 Ti 12 GB GPU.

4.1 Implementation Details

To make a fair comparison, for all ER algorithms, each off-policy algorithm is implemented with identical hyper-parameters.

Hyper-parameters. For deep actor-critic algorithms (SAC, TD3 and DDPG), both policy-network and value-network are represented using MLP with two hidden layers (256, 256) and optimized using Adam (Kingma and Ba 2014) with learning rare of 3×10^{-4}. The replay buffer size is 10^6 and the sampling mini-batch size is 256. The agents are evaluated every 5000 steps, by running the policy deterministically and cumulative rewards are averaged over 50 evaluation episodes. For DQN, the neural network is the same as in its original paper (Mnih et al. 2015), but we use the Adam optimizer instead of RMSProp to update the parameters. The learning rate is set to 2.5×10^{-4}. The replay buffer size and the sampling mini-batch size are 10^6 and 32, respectively. The agents are evaluated using the mean cumulative rewards of the last 100 episodes, every 2000 steps. For other hyper-parameters, we the use the same settings as in their original papers (Mnih et al. 2013; 2015; Haarnoja et al. 2018; Fujimoto, van Hoof, and Meger 2018; Lillicrap et al. 2016).

Prioritized experience replay. We implement the proportional variant of PER (Schaul et al. 2016). For DQN and DDPG, there is only one value-network that outputs $Q(s, a; \theta)$. The magnitude of temporal-difference (TD) error $|\delta|$ of the transition (s, a, r, s') is calculated as follow:

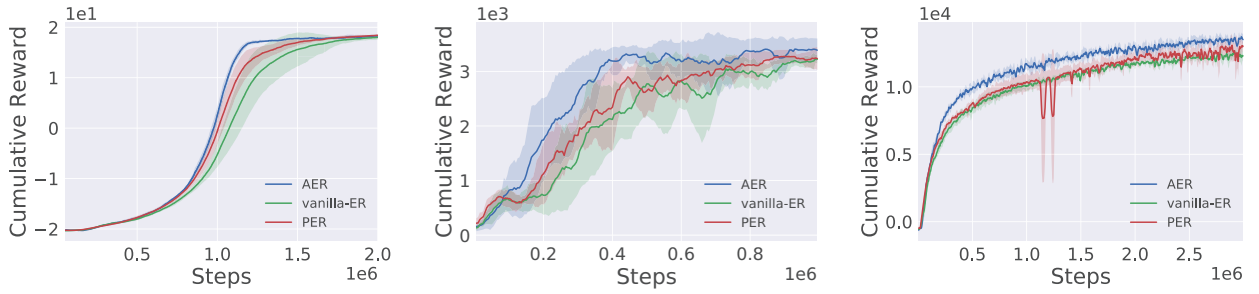
$$|\delta| = \left| r + \gamma Q\left(s', a'; \theta^-\right) - Q\left(s, a; \theta\right) \right|, \qquad (14)$$

where a' is the action given by current policy.

For TD3 and SAC, since they have two value-networks, we define the magnitude of TD error $|\delta|$ of each transition as the mean TD error of two value-networks:

$$|\delta| = \frac{1}{2} \sum_{j=1}^{2} \left| Q_{\text{backup}} - Q\left(s, a; \theta_j\right) \right|, \qquad (15)$$

where for SAC, $Q_{\text{backup}} = r + \gamma V(s'; \psi^-)$; for TD3, $Q_{\text{backup}} = r + \gamma \min_{i=1,2} Q\left(s', a'; \theta_i^-\right)$.



(a) Performanc of DQN on Pong (b) Performance of TD3 on Hopper (c) Performance of DDPG on HalfCheetah

Figure 4: Learning curves for DQN, TD3 and DDPG with AER, vanilla-ER and PER. DQN is trained on Pong; TD3 is trained on Hopper, and DDPG is trained on HalfCheetah

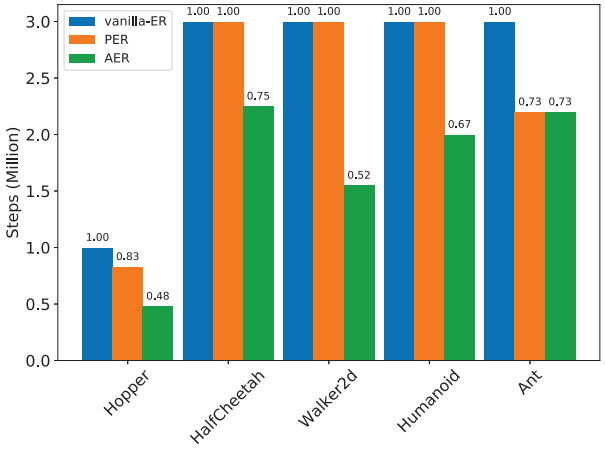


Figure 5: Total steps the learning agents take to achieve vanilla-ER's final performance. The normalized steps are plotted above the bars. Since *Swimmer* takes a much smaller number of steps to converge than other tasks, we omit it for the compactness of the plot.

Attentive experience replay. AER has two hyper-parameters: λ_0 and α. We did a coarse grid-search over the ranges: $\lambda_0 \in [2, 2.5, 3, 4], \alpha \in [0.5, 2/3, 1]$. For relatively simple tasks (Hopper-v2 and Swimmer-v2), we use $\lambda_0 = 4, \alpha = 1$; for the other complex tasks, we use $\lambda_0 = 2.5, \alpha = 2/3$. Besides these two hyper-parameters, AER also requires a function $\mathcal{F}(s_1, s_2)$ to measure the similarity between states s_1 and s_2. Since the state vectors are on different scales in MuJoCo tasks, we use the cosine similarity: $\mathcal{F}(s_1, s_2) = \frac{s_1 \cdot s_2}{\|s_1\|\|s_2\|}$. For Atari 2600 games, the raw states have extremely high dimensions and forbid direct similarity computing. We use a fixed and randomly initialized deep convolutional neural network, with the same architecture as the deep Q-network, to embed states into 512-dimensional features $x = \phi(s)$ (Burda et al. 2019a). Then, the similarity is defined as: $\mathcal{F}(s_1, s_2) = -\|\phi(s_1) - \phi(s_2)\|_2$.

4.2 Results for SAC

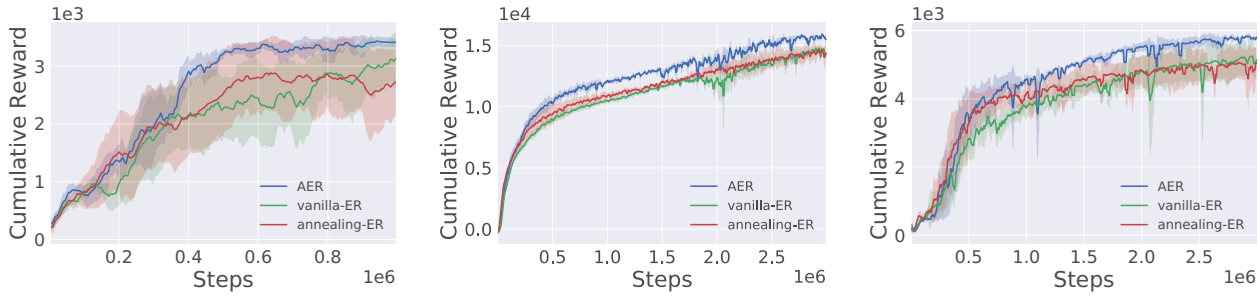
In this section, we couple AER, PER and vanilla-ER with SAC, a state-of-the-art off-policy actor-critic algorithm. The learning curves on 6 MuJoCo tasks are shown in Figure 3. Note that our implementation of SAC has comparable or even better performances compared to the results reported in the original paper (Haarnoja et al. 2018).

Figure 3 shows that the proposed AER algorithm consistently outperforms all the other ER methods on most of the tasks in terms of sample efficiency and final performance. On Ant-v2 and Swimmer-v2, all three methods converge to the optimal policy, but AER converges with clearly faster speed. On other tasks, higher sample efficiency and better final performance are observed. To give a more intuitive illustration, we calculate the total steps three agents take to reach the vanilla-ER's final performances. As shown in Figure 5, AER learns very efficiently and takes less than 65% of vanilla-ER's total steps to achieve the same performances.

It is worth noting that PER makes only a slight improvement to SAC. This phenomenon was also observed in previous works (Novati and Koumoutsakos 2019; Zha et al. 2019), when coupling PER with DDPG. We provide one possible explanation here. The PER method focuses on choosing informative transitions with high magnitude of TD error for the training of value-networks. The TD errors are calculated using value-networks, and therefore, transitions with high TD errors are more "surprising" to value-networks. But when applying PER to actor-critic algorithms, the sampled transitions are also used to update the policy-network. However, transitions with high TD errors usually diverge far from current policy and do harm to the updates of policy-network (Novati and Koumoutsakos 2019). On the contrary, AER selects transitions according to their similarities with current state. This preference to on-distribution states enforces transitions that contain old states to be discarded and stabilizes the training process of the policy-network.

4.3 Scalability of AER

In this section, we conduct experiments to show the scalability of different ER methods. We couple AER, PER and



(a) Hopper (b) HalfCheetah (c) Walker2d

Figure 6: Learning curves for SAC with AER, vanilla-ER and vanilla-ER with learning rate annealing.

vanilla-ER with three different off-policy algorithms on several tasks. The results are shown in Figure 4. AER shows consistent improvements over vanilla-ER and outperforms PER on different tasks. This observation indicates that on-distribution transitions are important for the training of both Q-learning and actor-critic learning. By contrast, PER outperforms vanilla-ER with clear margin only on DQN and has almost the same performances as vanilla-ER on TD3 and DDPG. This result verifies the our speculation: high TD error transitions are useful for the training of Q-network, but may hurt the performance of the policy-network.

4.4 The Effect of Learning Rate Annealing

In AER, we propose to anneal the learning rate to compensate the small gradients incurred by sampling small TD error transitions. But as known to us, in supervised learning with deep neural network, it is usually helpful to anneal the learning rate over time. With a high learning rate, the neural network converges faster, but with a small learning rate, the neural network converges with better performance. Hence, annealing the learning rate speeds up the learning of neural network in supervised learning. However, in deep reinforcement learning domain, faster weight-changes of the neural network incurred by high learning rate also forces the agent's policy to change more drastically, which can make the learning process unstable. Hence, annealing the learning rate does not necessarily improve the learning efficiency in deep RL algorithms.

To isolate the effect of learning rate annealing in AER, we combined vanilla-ER with learning rate annealing to form a new ER method: annealing-ER. Annealing-ER anneals the learning rate over time as AER does, but samples transitions uniformly. We couple vanilla-ER, annealing-ER and AER with SAC and test on three MuJoCo tasks. The results are plotted in Figure 6. Note that annealing-ER achieves nearly the same performance as vanilla-ER. This observation suggests that AER improves the learning efficiency due to the attentive sampling rather than learning rate annealing.

4.5 Running Time Comparison

We compare the running time of three different ER methods in Table 1. Since AER computes similarities only on the sampled batch of transition rather than the whole replay buffer, it requires negligible extra running time. On the other hand, PER needs nearly twice of the time to finish the same steps of training. This is due to the use of the "segment-tree" data structure to store transition priorities in PER. Searching in segment-tree needs extra $\mathcal{O}(\log N)$ time complexity, where N is the replay buffer size. Consequently, AER are also more efficient than PER in terms of running time.

	Running Time (10^3s)		
Env	vanilla-ER	AER	PER
Hopper	5.1	5.4	8
HalfCheetah	10.3	10.5	19
Walker2d	10.2	10.4	19
Swimmer	5.65	5.75	8.55
Humanoid	11	11.3	23.1
Ant	12.1	12.2	21

Table 1: Comparison of running time for vanilla-ER, AER and PER coupling with SAC.

5 Conclusion

In this paper, we study the problem of sampling transitions from a replay buffer for the training of off-policy algorithms. Motivated by the fact that each policy visits different states with different frequencies, we argue that transitions that contain states frequently visited by current policy are more important for the update of current policy. To this end, we introduce a novel experience replay algorithm AER. AER selects transitions according to the similarities between their states and the agent's current state. We couple AER with four off-policy algorithms and test on the suite of OpenAI Gym tasks. Experimental results suggest that AER consistently outperforms vanilla-ER and PER in terms of sample efficiency and final performance. AER shows that sampling on-distribution transitions for updates is a promising strategy.

Acknowledgements. This work was supported in part to Dr. Houqiang Li by NSFC under contract No. 61836011, and in part to Dr. Wengang Zhou by NSFC under contract No. 61822208 & 61632019 and Youth Innovation Promotion Association CAS (No. 2018497).

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 5048–5058.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; and Efros, A. A. 2019a. Large-scale study of curiosity-driven learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2019b. Exploration by random network distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

De Bruin, T.; Kober, J.; Tuyls, K.; and Babuška, R. 2015. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NeurIPS*.

De Bruin, T.; Kober, J.; Tuyls, K.; and Babuška, R. 2016. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3947–3952. IEEE.

De Bruin, T.; Kober, J.; Tuyls, K.; and Babuška, R. 2018. Experience selection in deep reinforcement learning for control. *The Journal of Machine Learning Research (JMLR)*.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Hasselt, H. V. 2010. Double q-learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Isele, D., and Cosgun, A. 2018. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

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

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3-4):293–321.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Novati, G., and Koumoutsakos, P. 2019. Remember and forget for experience replay. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5026–5033. IEEE.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2016. Sample efficient actor-critic with experience replay. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zha, D.; Lai, K.-H.; Zhou, K.; and Hu, X. 2019. Experience replay optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.