Journal of Artificial Intelligence Research 78 (2023) 639-672 Submitted 04/2023; published 11/2023

Actor Prioritized Experience Replay

Baturay Saglam BATURAY.SAGLAM@YALE.EDU
Department of Electrical Engineering
Yale University
New Haven, CT, USA

Furkan B. Mutlu BURAK.MUTLU@EE.BILKENT.EDU.TR
Dogan C. Cicek CICEK@EE.BILKENT.EDU.TR
Suleyman S. Kozat KOZAT@EE.BILKENT.EDU.TR
Department of Electrical and Electronics Engineering
Bilkent University
Ankara, Turkey

Abstract

A widely-studied deep reinforcement learning (RL) technique known as Prioritized Experience Replay (PER) allows agents to learn from transitions sampled with non-uniform probability proportional to their temporal-difference (TD) error. Although it has been shown that PER is one of the most crucial components for the overall performance of deep RL methods in discrete action domains, many empirical studies indicate that it considerably underperforms off-policy actor-critic algorithms. We theoretically show that actor networks cannot be effectively trained with transitions that have large TD errors. As a result, the approximate policy gradient computed under the Q-network diverges from the actual gradient computed under the optimal Q-function. Motivated by this, we introduce a novel experience replay sampling framework for actor-critic methods, which also regards issues with stability and recent findings behind the poor empirical performance of PER. The introduced algorithm suggests a new branch of improvements to PER and schedules effective and efficient training for both actor and critic networks. An extensive set of experiments verifies our theoretical findings, showing that our method outperforms competing approaches and achieves state-of-the-art results over the standard off-policy actor-critic algorithms.

1. Introduction

In off-policy deep reinforcement learning (RL), the experience replay buffer (ji Lin, 1992), which contains experiences that different policies may collect, is a vital ingredient of policy optimization (Lazaridis et al., 2020). Experience replay can stabilize and improve policy optimization by storing many previous experiences (or transitions) in a buffer and reusing them multiple times to perform gradient steps on policies and value functions approximated by deep neural networks (Sutton & Barto, 2018). Although the initial proposal of experience replay considered uniform sampling from the buffer, various sampling methods were shown to improve the data efficiency by calculating priority scores for the experiences (Schaul et al., 2015; Oh et al., 2021, 2022).

The use of priority-based non-uniform sampling in deep RL stems from a technique known as Prioritized Experience Replay (PER) (Schaul et al., 2015), in which high error transitions are sampled with higher likelihood, allowing for faster learning and reward propagation by focusing on the most crucial data. In fact, Hessel et al. (2018) demonstrated in an ablation study that the performance of the Deep Q-Network (DQN) algorithm (Mnih et al., 2015) was most significantly improved by the presence of PER, compared to other enhancements. Although the motivation of PER is intuitive for learning in discrete action spaces, e.g., The Arcade Learning Environment (Bellemare et al., 2013), many empirical studies showed that it substantially decreases the performance of off-policy actor-critic methods in continuous action domains, resulting in suboptimal or random behavior (Fujimoto et al., 2020; Oh et al., 2021, 2022). Unfortunately, the poor performance of PER in actor-critic lacks a critical theoretical foundation. In this study, we develop an analysis that provides insights into why off-policy actor-critic methods for the control of continuous systems are ineffective when combined with PER. In addition to this analysis, we propose novel modifications to PER to improve the empirical performance of the algorithm.

In continuous action spaces, the critic cannot be used to select actions due to the intractable, i.e., infinitely many possible actions (Sutton & Barto, 2018). Actor-critic methods can overcome this by employing a separate function, called the *actor*, to choose actions on the observed states. When combined with PER, the actor and critic networks are trained with transitions corresponding to large temporal-difference (TD) errors. TD error is a reasonable proxy that stands as the loss of the critic in Q-learning (Watkins & Dayan, 1992) and indicates the uncertainty and knowledge of the critic on the collected transitions in terms of the bootstrapped expected future rewards (Moore & Atkeson, 1993). Hence, in the basis of the TD-learning (Sutton, 1988), a large TD error implies that the critic has little knowledge and high uncertainty about the experiences (Sutton & Barto, 2018). However, we argue that actors cannot be effectively trained with experiences that the critic does not know their future returns well. An intuitive analogy may be that it is infeasible to expect a student to learn a subject well if the teacher has little knowledge about it. Our main theoretical contributions in this work justify our hypothesis that if an actor-critic algorithm is trained with a transition corresponding to a large TD error, the approximate policy gradient, i.e., computed under the Q-network, can significantly diverge from the actual gradient, i.e., computed under the optimal Q-function, for the transition of interest or the subsequent transition. This finding can be used to improve the performance of PER by training the actor with different experiences and facilitating the design of novel prioritization methods. Discoveries of this study can be summarized as follows:

- **Actor networks should be trained with low TD error transitions:** The critical implication of this finding is that the policy gradient, either stochastic or deterministic, depends on the critic. Therefore, it cannot be effectively computed using transitions on which the critic has high uncertainty. In particular, we find that a large TD error can correspond to a high Q-value estimation error for some transitions. Such transitions further causes the approximate policy gradient to diverge from the actual gradient under the optimal Q-function. To the best of our knowledge, this is the primary reason behind the poor performance of PER in standard off-policy actor-critic algorithms and we are the first to show it theoretically. Ultimately, this can only be overcome when the actor is trained with low TD error transitions in the TD error based prioritized sampling.

- **Actor and critic networks should be optimized with uniformly sampled transitions for a fraction of the batch size:** Training the actor and critic with completely different transitions, such as low and high TD error, is in violation of the actor-critic theory (Konda & Tsitsiklis, 1999). The interdependence between the critic and actor parameters is a fundamental tenet of this theory, as the actor's selected actions are incorporated into the updates and consequently influence the critic's parameters. Our empirical studies show that using a set of uniformly sampled transitions for a fraction of the batch size is extremely important in actor-critic training and ensures stability in learning.

- **Loss functions should be modified to prevent outlier bias leakage in the prioritized sampling:** While the combination of the latter two findings is theoretically and empirically favorable, prioritized and uniform sampling should not be considered in isolation from the loss function as outlier and biased transitions may still leak during sampling (Fujimoto et al., 2020). Notably, we demonstrate that corrections to PER cannot reach their maximum potential unless the mean-squared error (MSE) in the Q-network training is corrected. Thus, we leverage the prominent results of Fujimoto et al. (2020) in our modifications to PER.

In this paper, we introduce a novel experience replay prioritization framework, the Loss-Adjusted Approximate Actor Prioritized Experience Replay (LA3P) algorithm, to overcome the mentioned limitations of PER in off-policy actor-critic methods. LA3P effectively adapts PER to actor-critic algorithms by training the actor network with transitions that the critic has reliable knowledge of. Moreover, our algorithm considers the issues with stability and traditional actor-critic theory by not completely separating the actor and critic networks and adjusting the loss function accordingly. We assess the performance of LA3P on challenging continuous control benchmarks from OpenAI Gym (Brockman et al., 2016). Our results indicate that the introduced framework significantly outperforms the competing PER-correction algorithms by a wide margin, and achieves noteworthy gains over both PER and state-of-the-art methods in most of the domains tested. Furthermore, an extensive set of ablation studies verifies that each proposed modification to PER is essential to maintain the overall performance for actor-critic algorithms. All of our code and results are open-sourced and provided in the GitHub repository[1].

2. Related Work

Prioritized sweeping (Moore & Atkeson, 1993; Andre et al., 1997) for value iteration has been proposed as a method to enhance the learning speed and optimize computational resources in the initial studies of experience prioritization. Prioritized sweeping is a model-based reinforcement learning approach that attempts to focus an agent's limited computing resources to get a reasonable analysis of the environment's state values. It is also employed in modern RL applications to perform importance sampling over the collected trajectories (Schlegel et al., 2019) and learning from demonstrations (Hester et al., 2018).

Prioritized Experience Replay, which we investigate in depth in later sections, has been one of the most remarkable improvements to the DQN algorithm and its derivatives. Furthermore, it has been widely adopted in various learning algorithms, along with other notable improvements, such as Rainbow (Hessel et al., 2018), distributed PER (Horgan

[1] `https://github.com/baturaysaglam/LA3P`

et al., 2018), and distributional policy gradients (Barth-Maron et al., 2018). Modifications to PER have also been proposed, e.g., prioritizing the sequences of transitions (Gruslys et al., 2018) or optimization of the prioritization scheme (Zha et al., 2019). A counterpart to the mentioned experience replay approaches is determining which transitions to favor or forget (Novati & Koumoutsakos, 2019). Moreover, the effects originating from the composition and size of the experience replay buffers have also been studied by Isele and Cosgun (2018) and Liu and Zou (2018). The discussed methods usually consider model-based learning in discrete action domains, aiming at which sources to focus on or which features to select and store. Conversely, we focus on the challenges that arise from prioritized sampling in off-policy actor-critic model-free deep RL. Our aim is to devise an approach for determining which experiences should be replayed or sampled by using the prioritization scheme within the PER framework.

Lately, the learning-based approaches through deep function approximators have been proposed to determine which experiences to sample, independent of the experience replay buffer composition and without deciding which transitions to store. Zha et al. (2019) train a multi-layer perceptron whose input is the concatenation of reward, time step, and TD error. The network outputs a Bernoulli distribution to assign priorities to each sample within the replay buffer. On top of the neural sampler proposed by Zha et al. (2019), Oh et al. (2021) also considers TD error for prioritization, similar to PER. To train the sampler network, they employ the REINFORCE trick (Williams, 1992) that measures the increase in evaluation rewards achieved when the agent is trained with the priorities generated by the sampler network. The sampler network is then updated accordingly. In contrast to these learnable sampling strategies, we introduce corrections to the Prioritized Experience Replay in a rule-based manner.

Recently, corrections to PER have been extensively investigated by Fujimoto et al. (2020) and Oh et al. (2022). First, Fujimoto et al. (2020) addressed that any loss function combined with non-uniform probability may be converted into a new uniformly sampled loss function counterpart with the same expected gradient. By using this finding, they completely replaces the loss in PER with a modified loss function, which has been shown to have no effect on empirical performance. By correcting its uniformly sampled loss function equivalent, this connection provides a new branch of PER improvements called Loss Adjusted Prioritized (LAP) Experience Replay (Fujimoto et al., 2020). We broadly investigate LAP in the later sections. Secondly, Oh et al. (2022) argue that sampling from the replay buffer depending highly on the TD error (or Q-network's error) may be ineffective due to the under- or overestimation of the Q-values resulting from the deep function approximators and bootstrapping. For this reason, they proposed to learn auxiliary features driven from the components in model-based RL to calculate the scores of experiences. The proposed method, Model-Augmented PER (MaPER) (Oh et al., 2022), uses the critic network to improve the effect of curriculum learning for predicting Q-values with minimal memory and computational overhead compared to vanilla PER. Ultimately, we include the theoretical results of Fujimoto et al. (2020) in our methodology, while we also compare our method against uniform sampling, PER, LAP, and MaPER in our empirical studies.

3. Technical Preliminaries

We first examine the general deep reinforcement framework considered in this study. Next, we provide some technical aspects of the Prioritized Experience Replay algorithm required to establish the methodology.

3.1 Deep Reinforcement Learning

This study considers the standard RL setup, described by Sutton and Barto (2018) and Kaelbling, Littman, and Moore (1996). At every discrete time step t, the agent observes a state s and chooses an action a. Depending on its action selection, the agent receives a reward r and observes the next state s'. The RL problem is often framed by a Markov decision process consisting of a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, a reward function $\mathcal{R}$, the environment dynamics model P, and the discount factor $\gamma \in [0, 1)$.

The performance of a policy π is assessed under the action-value function (Q-function or critic) Q^{π}, which represents the expected sum of discounted rewards while following the policy π after performing the action a in state s:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^{t} r_{t+1} | s_0 = s, a_0 = a].$$

The action-value function is determined through the Bellman equation (Bellman, 2003):

$$Q^{\pi}(s, a) = \mathbb{E}_{r, s' \sim P, a' \sim \pi}[r + \gamma Q^{\pi}(s', a')],$$

where a' is the next action selected by the policy on the observed next state s'.

In deep RL, the critic is approximated by a deep neural network Q_{θ} with parameters θ. Then, the Q-learning with deep function approximators transforms into the standard DQN algorithm. Given a transition tuple $\tau = (s, a, r, s')$, DQN is trained by minimizing the loss $\delta_{\theta}(\tau)$ based on the temporal-difference error corresponding to the Q-network Q_{θ}. The TD-learning algorithm (Sutton, 1988) is an update rule based on the Bellman equation, which defines a fundamental relationship used to learn the action-value function by bootstrapping from the value estimate of the current state-action pair (s, a) to the subsequent state-action pair (s', a'):

$$\begin{aligned} y(\tau) &= r + \gamma Q_{\theta'}(s', a'), \\ \delta_{\theta}(\tau) &= y(\tau) - Q_{\theta}(\tau). \end{aligned}$$

Note that TD error in the latter equation can also be regarded as $\delta_{\theta}(\tau) = Q_{\theta}(\tau) - y(\tau)$, however, it does not affect our analysis. Transitions $\tau \in \mathcal{B}$ are sampled through a sampling method from the experience replay buffer that contains a previously collected set of experiences in the form of a batch of transitions $\mathcal{B}$. The target $y(\tau)$ in (3.1) employs a separate target network with parameters θ' that maintains stability and fixed objective in learning the optimal Q-function. The target parameters are updated either with the soft or hard update by copying parameters θ from the Q-network Q_{θ} by a small fraction ζ at every step or to match θ every fixed period, respectively. In each update step, the loss for the Q-network is averaged over the sampled batch of transitions $\mathcal{B}$: $\frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \delta_{\theta}(\tau)$, where $|\mathcal{B}|$ is the number of transitions contained in $\mathcal{B}$. Note that the batch size does not affect our analysis on the prioritization as the expected gradient remains unchanged.

In controlling continuous systems (i.e., continuous action domains), the maximum $\max_{\tilde{a}} Q(s, \tilde{a})$ to select actions is intractable due to an infinite number of possible actions. To overcome this issue, actor-critic methods employ a distinct network to represent the policy π_ϕ, parameterized by ϕ, to determine the actions based on the observed states. The policy network is commonly referred to as the actor network, and it can be either deterministic $a = \pi_\phi(s)$ or stochastic $a \sim \pi_\phi(\cdot|s)$. The next action a' in the construction of the Q-network target in (3.1) can be chosen either by the behavioral actor network π_ϕ (i.e., the policy that the agent uses for action selection) or a distinct target actor network $\pi_{\phi'}$, depending on the actor-critic method. Equivalent to the target Q-network, the target actor network with parameters ϕ' is used to ensure stability over the updates. Finally, the policy is optimized with respect to the policy gradient $\nabla\phi$ computed by a policy gradient technique, the loss of which is explicitly or implicitly based on maximizing the Q-value estimate of Q_θ. Additionally, the Q-network is trained through the DQN algorithm or one of its variants such as the Clipped Double Q-learning algorithm (Fujimoto et al., 2018).

3.2 Prioritized Experience Replay

Prioritized Experience Replay is a non-uniform sampling strategy for replay buffers in which transitions are sampled in direct proportion to their absolute TD error. The primary reasoning for PER is that training on the highest error samples will yield the most significant performance improvement. PER introduces two modifications over the standard uniform sampling. First, a stochastic prioritization scheme is used. The motivation is that TD errors are updated only for replayed transitions. As a result, the initially high TD error transitions are updated more frequently, resulting in a greedy prioritization. Also, the noisy Q-value estimates increase the variance due to the greedy sampling. Therefore, overfitting is inevitable if one directly samples transitions proportional to their TD errors. To remedy this, a probability value is assigned to each transition τ_i, proportional to its absolute TD error $|\delta_\theta(\tau_i)|$, and set to the power of a hyperparameter α to smooth out the extremes:

$$p(\tau_i) = \frac{|\delta_\theta(\tau_i)|^\alpha + \mu}{\sum_j (|\delta_\theta(\tau_j)|^\alpha + \mu)}, \tag{1}$$

where a small constant μ is added to avoid assigning zero probabilities to transitions. Otherwise, they would not be sampled again. This is required since the most recent value of a transition's TD error is approximated by the TD error when it was last sampled.

Second, favoring large TD error transitions with the stochastic prioritization shifts the distribution of s' to $\mathbb{E}_{s'}[Q(s', a')]$. This can be corrected through importance sampling with ratios $w(\tau_i)$:

$$\hat{w}(\tau_i) = \left(\frac{1}{|R|} \cdot \frac{1}{p(\tau_i)}\right)^\beta,$$
$$w(\tau_i) = \frac{\hat{w}(\tau_i)}{\max_j \hat{w}(\tau_j)},$$
$$\mathcal{L}_{\text{PER}}(\delta_\theta(\tau_i)) = w(\tau_i)\mathcal{L}(\delta_\theta(\tau_i)), \tag{2}$$

where $\mathcal{L}$ indicates the TD error under prioritization, $|R|$ is the total number of transitions in the replay buffer, and the hyperparameter β is used to smooth out the high variance induced by the importance sampling weights. The latter equation corrects the distribution shift by employing a ratio between uniform sampling with probability $\frac{1}{|R|}$ and the ratio specified in (1). This approach reduces the impact of high priorities on the distribution. At last, the β value is annealed from a predefined initial value β_0 to 1, to eliminate the bias introduced by the distributional shift.

4. Prioritized Sampling in Actor-Critic Algorithms

We start by building the theoretical foundations for the performance degradation of the prioritized sampling in actor-critic methods. In Assumption 1, we first demonstrate that there may exist transitions such that the absolute value of the associated TD error can increase the absolute Q-value estimation error. Then, using our theoretical implications, we prove in Theorem 1 that if a policy is optimized using transitions that exhibit large TD error, the gradient of the approximate actor network computed using the Q-network's estimates will deviate from the actual gradient computed under the optimal Q-function, leading to divergence. In addition to our theoretical investigation, we also tackle the issues highlighted in a prior study that shed light on the suboptimal performance of PER when used with standard off-policy actor-critic methods.

Assumption 1. *Consider the temporal-difference error δ_θ associated with the critic network Q_θ in off-policy actor-critic algorithms. Then, there exists a transition tuple $\tau_i = (s_i, a_i, r_i, s_{i+1})$ with $\delta_\theta(\tau_i) \neq 0$ such that if the absolute temporal-difference error on τ_i increases, the absolute Q-value estimation error on at least τ_i or τ_{i+1} will also increase.*

Discussion of Assumption 1 To simplify the analysis, we consider that the target networks do not affect the estimation, as they aim to ensure stability and a fixed objective over the updates (Mnih et al., 2015). We begin by expanding the temporal-difference error $\delta_\theta(\tau_i)$ in terms of the bootstrapped value estimation:

$$\delta_\theta(\tau_i) = r_i + \gamma Q_\theta(s_{i+1}, a_{i+1}) - Q_\theta(s_i, a_i), \tag{3}$$

where $a_{i+1} \sim \pi_\phi(s_{i+1})$ represents the action selected by the policy network π_ϕ on the observed next state s_{i+1}. We note that the optimal action-value function Q^π under the policy π yields no TD error:

$$\delta^\pi(\tau_i) = r_i + \gamma Q^\pi(s_{i+1}, a_{i+1}) - Q^\pi(s_i, a_i) = 0. \tag{4}$$

Then, subtracting (4) from (3) results in

$$\delta_\theta(\tau_i) = \gamma \underbrace{\left(Q_\theta(s_{i+1}, a_{i+1}) - Q^\pi(s_{i+1}, a_{i+1})\right)}_{:=\epsilon_{\tau_{i+1}}} - \underbrace{\left(Q_\theta(s_i, a_i) - Q^\pi(s_i, a_i)\right)}_{:=\epsilon_{\tau_i}} \neq 0.$$

Naturally, ϵ_{τ_i} and $\epsilon_{\tau_{i+1}}$ represent the estimation error at the current and subsequent steps, respectively. We now consider the following cases for different signs of these variables when the absolute TD error increases, and show that it is possible that if the absolute value of TD error increases, then the absolute value of at least ϵ_{τ_i} and $\epsilon_{\tau_{i+1}}$ also increases. Note that $\gamma \geq 0$ is omitted as it is fixed.

Case 1: $|\delta_\theta(\tau_i)|$ increases when $\delta_\theta(\tau_i), \epsilon_{\tau_i}, \epsilon_{\tau_{i+1}} > 0$. This can happen if $\epsilon_{\tau_{i+1}}$ becomes a greater positive number as well, which implies that $|\epsilon_{\tau_{i+1}}|$ eventually increases.

Case 2: $|\delta_\theta(\tau_i)|$ increases when $\delta_\theta(\tau_i), \epsilon_{\tau_{i+1}} > 0$ and $\epsilon_{\tau_i} < 0$. This can be achieved as long as either $\epsilon_{\tau_{i+1}}$ increases or ϵ_{τ_i} becomes a smaller negative number. Eventually, at least one of $|\epsilon_{\tau_{i+1}}|$ and $|\epsilon_{\tau_i}|$ will increase.

Case 3: $|\delta_\theta(\tau_i)|$ increases when $\delta_\theta(\tau_i), \epsilon_{\tau_i} > 0$ and $\epsilon_{\tau_{i+1}} < 0$. This case is impossible to occur as $\delta_\theta(\tau_i)$ is positive initially.

Case 4: $|\delta_\theta(\tau_i)|$ increases when $\delta_\theta(\tau_i) > 0$ and $\epsilon_{\tau_i}, \epsilon_{\tau_{i+1}} < 0$. If both estimation error terms are negative and $\delta_\theta(\tau_i)$ is positive initially, it means that the magnitude of ϵ_{τ_i} is larger than that of $\epsilon_{\tau_{i+1}}$. If $\delta_\theta(\tau_i)$ increases to a greater positive number, ϵ_{τ_i} may become a smaller negative number. Therefore, $|\epsilon_{\tau_i}|$ will increase as well.

The remaining cases: $|\delta_\theta(\tau_i)|$ increases when $\delta_\theta(\tau_i) < 0$. For the remaining cases where $\delta_\theta(\tau_i)$ becomes a smaller negative number, we will reach the same conclusion as in the first four cases if we multiply both sides with the minus sign. Therefore, the absolute value of at least ϵ_{τ_i} or $\epsilon_{\tau_{i+1}}$ can still increase.

It is important to note that this behavior may not always lead to the deduction proved. However, we infer that there is always a possibility for it to occur under the presence of a TD error with an increasing magnitude, which is the primary proxy for the prioritized sampling of interest.

Our assumption is reasonable and likely to occur in practice, as it is based on the fundamental principles of off-policy actor-critic reinforcement learning. In particular, the TD error is a key factor in determining the update rule for the critic network (Sutton & Barto, 2018), being the difference between the expected value of the current state-action pair and the actual observed value. TD error can arise due to various reasons such as errors in the policy or the approximate Q-function. When the critic network is updated to reduce the TD error, it also affects the estimation error since the agent's Q-value estimates are based on the TD error. A large TD error implies that the predicted Q-value is far from the true value, which in turn suggests that the Q-value estimation error is likely to be high as well (Sutton & Barto, 2018, Chapter 6).

Moreover, in off-policy learning, the Q-value estimation error is influenced by the difference between the behavior policy and the target policy. If the TD error is large, it may indicate that the difference between the policies is significant, making it more likely that the Q-value estimation error will also be large. Therefore, our assumption not only reflects the practical challenges that an agent may face but is also supported by the RL literature. For instance, Schaul et al. (2015) and Sutton and Barto (2018) discusses the relationship between TD error and Q-value estimation error, stating that a large absolute TD error indicates poor prediction and a need for stronger weight updates in the direction of the error. We now leverage our latter result to explain why policy optimization cannot be effectively performed using transitions with large TD errors, following the assumption made. Specifically, Theorem 1 highlights the importance of transitions with minimal TD errors during policy optimization, as transitions with large errors can lead to suboptimal policies and cause the approximate policy gradient to diverge from its true value.

Theorem 1. *Let τ_i be a transition such that Assumption 1 is satisfied. Then, if the absolute temporal-difference error at time step i increases, the computed policy gradient will diverge from the true policy gradient for at least the current step i or the subsequent step $i+1$.*

Proof. The proof relies on the policy gradient theorem of Sutton et al. (2000) in the deep function approximation setting. To begin, we formally expresses the standard policy iteration with function approximation in terms of the policy parameters ϕ:

$$\phi \leftarrow \phi + \eta \nabla \phi,$$

where $\nabla \phi$ is the computed policy gradient, and η is the learning rate. A general formulation for the policy gradient with function approximation is given by

$$\nabla \phi \coloneqq \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \phi} Q_{\theta}(s,a),$$

where Q_θ is the function approximation of Q^π, and $d^\pi(s)$ is a discounted weighting of states encountered, starting from s_0 and then following π: $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s | s_0, \pi)$. In the latter equation, the gradient is computed by taking the average over the continuous state and action spaces. However, computing the gradient over the entire state and action space is often computationally infeasible. Instead, the gradient can be estimated over a subset of the state and action space, which is equivalent to the sampled mini-batch of transitions in our case. For simplicity, we consider one-step gradient computation and omit computing the gradient over the sampled mini-batch of transitions. For the state-action pair $(s_i, a_i) \in \tau_i$, the policy gradient $\nabla \phi(\tau_i)$ is proportional to $d^\pi(s_i)$ multiplied by the gradient of $\pi(s_i, a_i)$ with respect to ϕ and weighted by the estimated Q-value of (s_i, a_i). Let $k \coloneqq d^\pi(s_i) \frac{\partial \pi(s_i, a_i)}{\partial \phi}$, and we have

$$\begin{aligned} \nabla \phi(\tau_i) &= k Q_\theta(s_i, a_i), \\ \nabla \phi_{\text{true}}(\tau_i) &= k Q^\pi(s_i, a_i), \end{aligned}$$

for the approximated and true policy gradients, respectively. The same k applies to both the computed and true policy gradients since it is independent of the Q-value estimates. Additionally, since k is constant over the range of values of the Q-value estimates, we can treat it as a constant for the proportionality between the policy gradients and Q-value estimates. As defined in Assumption 1, we have $Q_\theta(s_i, a_i) = Q^\pi(s_i, a_i) + \epsilon_{\tau_i}$. Using this, we can write

$$\begin{aligned} \nabla \phi(\tau_i) - \nabla \phi_{\text{true}}(\tau_i) &= k(Q_\theta(s_i, a_i) - Q^\pi(s_i, a_i)) \\ &= k(Q^\pi(s_i, a_i) + \epsilon_{\tau_i} - Q^\pi(s_i, a_i)) \\ &= k \epsilon_{\tau_i}. \end{aligned}$$

Taking the absolute value of the latter equation gives

$$|\nabla \phi(\tau_i) - \nabla \phi_{\text{true}}(\tau_i)| = |k \epsilon_{\tau_i}|, = |k||\epsilon_{\tau_i}| = k|\epsilon_{\tau_i}|.$$

Although the k term is not a constant, but rather a variable, we can still take it out of the absolute operator since it is a product of probabilities. Therefore, the equation above implies that the absolute difference between the computed and true policy gradients is proportional to the absolute Q-value estimation error:

$$|\nabla\phi(\tau_i) - \nabla\phi_{\text{true}}(\tau_i)| \propto |\epsilon_{\tau_i}|,$$

where we still use k as a constant of proportionality since it does not vary over the Q-value estimates. Considering that Assumption 1 is satisfied, an increase in the absolute TD error in the current step leads to a corresponding increase in the absolute estimation error in either the current or a subsequent step. The latter equation further suggests that an increase in the absolute estimation error leads to a larger discrepancy between the true and computed policy gradients in the corresponding step. Hence, we deduce that an increase in the absolute TD error causes the computed policy gradient to deviate from the true policy gradient, at least in the current or a subsequent step. $\square$

Theorem 1 states that if the actor network is optimized with a transition corresponding to a large TD error, the resulting policy gradient at the current or subsequent step can deviate from the true gradient. We formally state this finding in Corollary 1.

Corollary 1. *An increase in the absolute value of the temporal-difference error can lead to a less accurate approximate policy gradient, causing it to deviate from the true policy gradient corresponding to the optimal Q-function, at least for the current or subsequent transition.*

This forms an essential ingredient in the degraded performance of the prioritized sampling when an actor network is employed. We now address a recent finding that provides an explanation for the poor performance of prioritized sampling in off-policy actor-critic algorithms and complements our investigation. As previously mentioned, prioritizing transitions with high TD error via stochastic sampling leads to a shift in the distribution of s' to $\mathbb{E}_{s'}[Q(s', a')]$. Therefore, this induced bias is corrected by importance sampling expressed in (2). However, Fujimoto et al. (2020) argued that PER does not entirely eliminate the bias and may favor outliers when combined with MSE loss in the Q-network updates. Specifically, when MSE is used in combination with PER, optimizing it using a loss of $\frac{1}{\rho}|\delta_\theta(\tau)|^\rho$, where $\rho + \alpha - \alpha\beta \neq 2$, leads to bias in the Q-network target and, consequently, biased Q-network updates. Remark 1 underscores this finding, which was identified by Fujimoto et al. (2020) as one of the contributing factors to the suboptimal performance of PER when used in conjunction with standard actor-critic algorithms in continuous action domains.

Remark 1. *The PER objective is biased if $\rho + \alpha - \alpha\beta \neq 2$ under the loss function $\frac{1}{\rho}|\delta_\theta(\tau)|^\rho$ (Fujimoto et al., 2020).*

Based on Remark 1, Fujimoto et al. (2020) inferred that prioritized sampling is subject to bias when used with MSE because the condition $\rho + \alpha - \alpha\beta \neq 2$ is never precisely satisfied, i.e., if $\beta < 1$, then $2 + \alpha - \alpha\beta > 2$ for $\alpha \in (0, 1]$. Moreover, the induced bias is not always the same. On the contrary, an L1 loss can satisfy this property such that $1 + \alpha - \alpha\beta \in [1, 2]$ for $\alpha \in (0, 1]$ and $\beta \in [0, 1]$ since $\rho = 1$ in the L1 loss. In practice, however, the L1 loss may not be ideal since each update entails a constant-sized step, which may cause the objective to be overstepped if the learning rate is too large. To address this issue, Fujimoto et al. (2020)

opted to employ the Huber loss with $\kappa = 1$, a commonly used loss function, in the Q-network updates instead of the MSE loss:

$$\mathcal{L}_{\text{Huber}}(\delta_\theta(\tau_i)) = \begin{cases} 0.5\delta_\theta(\tau_i)^2 & \text{if } |\delta_\theta(\tau_i)| \leq \kappa, \\ |\delta_\theta(\tau_i)| & \text{otherwise.} \end{cases} \tag{5}$$

When the error is below threshold 1, the Huber loss swaps from L1 to MSE and properly scales the gradient as $\delta_\theta(\tau_i)$ approaches zero. When $|\delta_\theta(\tau_i)| < 1$, MSE is applied. Thus, samples with an error of less than 1 should be sampled uniformly to prevent the bias induced by MSE and prioritization. This is achieved in the LAP algorithm via the prioritization scheme, $p(\tau_i) = \max(|\delta_\theta(\tau_i)|^\alpha, 1)$, through which samples with low priority are clipped to be at least 1. The LAP algorithm resolves the issue outlined in Remark 1 by incorporating a modified stochastic prioritization scheme in combination with the Huber loss:

$$p(\tau_i) = \frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)}. \tag{6}$$

The results presented by Fujimoto et al. (2020) form a complement to our theoretical conclusions for the poor performance of PER, which we underline in Remark 2. We note that our emphasis is on the off-policy actor-critic algorithms in which actions are selected by a separate actor network trained to maximize the action-value estimate of the Q-network. In contrast, Remark 2 comprises both discrete control and off-policy actor-critic deep RL. There may be another justification for the poor performance of PER such as inaccurate Q-value estimates. Nevertheless, such reasons are not PER-dependent and are induced by the derivatives of the DQN algorithm. Therefore, to the best of our knowledge, Corollary 1 and Remark 2 establish the basis for the algorithmic drawbacks of PER in off-policy actor-critic.

Remark 2. *To further eliminate the bias favoring the outlier transitions in the Prioritized Experience Replay algorithm, the Huber loss with* $\kappa = 1$ *should be used in conjunction with the prioritization scheme expressed in* (6) *(Fujimoto et al., 2020).*

5. Adaptation of Prioritized Experience Replay to Actor-Critic Algorithms

First, we introduce a set of modifications to vanilla PER to address the issues induced by prioritized sampling in actor-critic algorithms. We then explain why these modifications cannot be directly applied and conclude with the proposed method.

5.1 Inverse Sampling for the Actor Network

We start with Corollary 1, which suggests that training the actor network with a large TD error transition can cause the approximate policy gradient to diverge from the actual gradient, at least for the current or subsequent transitions. However, it should be noted that if the policy gradient diverges only for subsequent transitions that do not correspond to a large TD error, the performance may not necessarily degrade under the TD error-based prioritization scheme. Nonetheless, this scenario is unlikely to occur, as the replay buffer typically contains only a few transitions in the initial optimization steps, which is typically smaller than the batch size.

Observation 1. *The performance of actor-critic methods may not degrade under the PER algorithm if the transitions subsequent to the sampled transitions are not used to optimize the actor network. Nevertheless, this remains a slight possibility in the standard off-policy actor-critic algorithms.*

To address the issue identified in 1, we propose optimizing the actor network using transitions that have small TD errors. To achieve this, we employ inverse sampling from the prioritized replay buffer, which involves sampling transitions with low TD errors for the actor network using the PER approach. To implement this approach, it is necessary to analyze the PER data structure. One efficient and popular implementation of PER, which corresponds to *proportional prioritization*, is based on a "sum-tree" data structure. This sum-tree data structure is very similar to the array representation of a binary heap, with the main difference being that instead of the standard heap property, a parent node's value is equal to the sum of its children. The internal nodes of the sum-tree data structure are intermediate sums, with the parent node carrying the sum over all priorities p_{total}. In the meantime, the leaf nodes store transition priorities. This allows for $\mathcal{O}(\log|R|)$ updates and sampling while calculating the cumulative total of priorities. For sampling a mini-batch of size N, the range $[0, p_{\text{total}}]$ is evenly divided into N ranges. Within each range, a value is uniformly sampled. The sum-tree data structure is then queried for the transitions that correspond to each sampled value.

One intuitive approach to sampling transitions with a probability inversely proportional to the TD error is to create a new sum-tree that contains the global inverse of the priorities. Although priorities in vanilla PER are updated for every training step through the previously defined sum tree data structure, inverse sampling for actor updates requires creating a new sum tree prior to training. In every update step, priorities are calculated using

$$I \sim \tilde{p}(\tau_i) = \frac{p_{\max}}{p(\tau_i)} = \max_i \left(\frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)} \right) \cdot \frac{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)}{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}, \tag{7}$$

where $p(\tau_i)$ is the priority of the i^{th} transition and $p_{\max}$ is the maximum of the previously determined priorities of the stored transitions. As highlighted in Remark Remark 2, the use of MSE with PER may still result in varying biases that could potentially favor outlier transitions. To address this concern, we adopt the prioritization scheme proposed by the LAP algorithm, expressed by (6), in (7). Notice that (7) does not alter proportional prioritization. The relative proportions (e.g., the largest over the smallest) do not change as we take the inverse by multiplication.

This forms the core component of our approach. To mitigate the impact of outlier bias also in the Q-network updates, we again adopt the Huber loss function with a tuning parameter of $\kappa = 1$, as defined in (5), similar to the LAP algorithm. Consequently, during each optimization step t, both the Q-network and priorities are updated respectively using

$$I \sim p(\tau_i) = \frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)},$$
$$\theta \leftarrow \theta - \eta \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\theta \mathcal{L}_{\text{Huber}}(\delta_\theta(\tau_i)),$$
$$p(\tau_i) \leftarrow \max(|\delta_\theta(\tau_i)|^\alpha, 1) \text{ for } i \in I,$$

where I denotes the indices of the prioritized transitions that are sampled to form the mini-batch. Note that the clipping reduces the likelihood of dead transitions when $p(\tau_i) \approx 0$, which eliminates the need for the μ parameter. It is important to note that the Huber loss function cannot be employed in the computation of the policy gradient due to several reasons. First, the priorities are determined by the loss function of the Q-network, which is the TD error, and the use of MSE loss in conjunction with TD error-based prioritized sampling is the primary cause of the mentioned outlier bias. Also, the policy loss and gradient are computed using a class of policy gradient techniques that cannot be substituted by the Huber loss. Therefore, the outlier bias does not affect the policy gradient, and there is no need to use the Huber loss in the policy network.

5.2 Optimizing the Actor and Critic with a Shared Set of Transitions

In some cases, optimizing the actor and critic networks with entirely different transitions can potentially violate the actor-critic theory. Typically, the features used by the critic network are dependent on the actor parameters and policy gradient since the actor determines the actions that ultimately lead to the observed state space (Konda & Tsitsiklis, 1999). An important corollary to this observation is that if the critic is updated using a set of features that exist in a state-action space where the actor is never optimized, this may lead to significant instability. This is because the transitions used by the critic are processed through the actor, and if the actor never sees these transitions, the critic's updates may not accurately reflect the actual performance of the actor (Konda & Tsitsiklis, 1999). Therefore, the action evaluations of the critic might become questionable.

Intuitively, this situation can occur when inverse prioritized and prioritized sampling are used for the actor and critic networks, respectively, since they may never be optimized with the same transitions. The reason for this is that the TD error of the critic's samples may not decrease to the extent that the actor is unable to observe them. We indicated that there is not always a direct correlation between TD and estimation errors. Hence, some of the transitions might initially have low TD errors. If the actor is optimized with respect to these low TD error transitions throughout the learning and the Q-network only focuses on the remaining large TD error transitions, samples used in the actor and critic training may not be the same. Although this remains unlikely, we nevertheless prevent it by updating the actor and critic networks through a set of shared transitions, being a fraction of the sampled mini-batch of transitions. However, we could not know the value of such a fraction, and we will introduce it as a hyperparameter later. We emphasize these deductions in Observation 2 and regulate our approach accordingly.

Observation 2. *If the transitions for the actor and critic are sampled through inverse prioritized and prioritized sampling, respectively, they may never observe the same transitions. This, in turn, violates the actor-critic theory as outlined by Konda and Tsitsiklis (1999) and can result in unstable learning. Hence, the actor and critic should be optimized with the same set of transitions, at least for a fraction of the sampled mini-batch of transitions, in every update step.*

How to choose the set of shared transitions? We inspect the following sampling alternatives in terms of the TD error. We can additionally investigate auxiliary variables driven by the components in model-based RL to compute the scores of the experiences, similar to the work of Oh et al. (2022). However, learning additional features introduces additional computational overhead. This aspect, however, is beyond the scope of this study, as our primary focus is solely on prioritization regarding the TD error and corrections to vanilla PER in off-policy actor-critic methods.

- **Transitions with large TD error:** The actor network cannot be optimized with experiences that have large TD error since the policy gradient significantly diverges from the actual gradient, as discussed in Corollary 1.
- **Transitions with small TD error:** Learning from the experiences with small TD error can be beneficial, yet they decrease the sampling efficiency and waste resources since the Q-network has little to learn from small TD error transitions in the sense of prioritized sampling.
- **Uniformly sampled transitions:** The latter two alternatives imply that uniform sampling for the set of shared transitions can be a suitable choice. Although large TD error transitions might be included in the uniformly sampled mini-batch, their effects are reduced due to averaging in the mini-batch learning.

Following the latter discussion, we determine that uniform sampling for a fraction of the transitions in the sampled mini-batch is a suitable approach for addressing the issue highlighted in Observation 2. While we could also consider using transitions with an average magnitude of TD errors, the use of uniform sampling already encompasses transitions with a mean TD error in the expectation. Furthermore, relying on random sampling allows for the inclusion of transitions that cannot be sampled by prioritized and inverse prioritized sampling. However, other distributions, such as those that encourage mean tendency/variance reduction, could be promising directions for future research.

As discussed in Remark 2, the combination of Huber loss ($\kappa = 1$) with the prioritized sampling can eliminate outlier bias in the LAP algorithm. Fujimoto et al. (2020) also introduced the mirrored loss function of LAP, with an equivalent expected gradient, for uniform sampling from the experience replay buffer. To observe the same benefits of LAP also in the uniform sampling counterpart, its mirrored loss function, Prioritized Approximate Loss (PAL), should be employed instead of MSE. Similar to the case of the LAP function, the PAL loss is also not employed in the policy network's updates. The PAL function is expressed by

$$\xi = \frac{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)}{N},$$
$$\mathcal{L}_{\text{PAL}}(\delta_\theta(\tau_i)) = \frac{1}{\xi}\begin{cases} 0.5\delta_\theta(\tau_i)^2 & \text{if } |\delta_\theta(\tau_i)| \leq 1, \\ \frac{|\delta_\theta(\tau_i)|^{1+\alpha}}{1+\alpha} & \text{otherwise.} \end{cases} \tag{8}$$

Remark 3. *To eliminate the outlier bias in the uniform sampling counterpart, the Prioritized Approximate Loss (PAL) function should be employed, which has the same expected gradient as the Huber loss when combined with PER (Fujimoto et al., 2020).*

Having the latter component included, this forms our PER correction algorithm, **L**oss-**A**djusted **A**pproximate **A**ctor **P**rioritized Experience Replay (LA3P). In summary, our approach involves uniformly sampling a mini-batch of transitions, with a size of $\lambda \cdot N$ in each update step. Here, $\lambda \in [0, 1]$ represents the fraction of the transitions that are uniformly sampled and serves as the only introduced hyperparameter in our approach. Using the uniformly sampled batch, the critic and actor networks are optimized consecutively. The critic is updated based on the PAL function expressed in (8), and the priorities are updated immediately after. Following this, a total of $(1 - \lambda) \cdot N$ transitions are sampled through prioritized and inverse prioritized sampling for the critic and actor networks, respectively. The critic is then optimized using the Huber loss ($\kappa = 1$) expressed in (5), while a policy gradient technique optimizes the actor. Lastly, the priorities are updated again. Overall, both the actor and critic networks are optimized with N transitions per update step, similar to standard off-policy actor-critic algorithms. Note that we update the priorities also in the uniform counterpart since the score of the transitions should be up-to-date whenever possible. In addition, the order of the prioritized and uniform updates does not alter the expected gradient. Hence, it does not matter which of them is used first. Nevertheless, in our implementation and experiments, we employ the structure outlined in Algorithm 1, denoted through a generic application to off-policy actor-critic methods. A clear summary of the LA3P framework is also depicted in Appendix A.

Ultimately, we conduct a complexity analysis for our approach. The LA3P framework introduces an additional sum tree, the LAP, and PAL functions on top of vanilla PER. As LAP and PAL operate on the sampled batches of transitions, the computational complexity introduced by these modifications is dominated by the additive sum tree, which operates on the entire replay buffer. Moreover, the additive sum tree requires a priority update. Setting the priorities of the nodes has the same complexity as in PER. Additionally, LA3P takes the inverse of the priorities by multiplication, which takes $\mathcal{O}(|R|)$ runtime. Therefore, LA3P operates in $\mathcal{O}(\log|R|) + \mathcal{O}(|R|)$. As $\mathcal{O}(|R|)$ dominates $\mathcal{O}(\log|R|)$, we conclude that LA3P has a runtime of $\mathcal{O}(|R|)$ in the worst case scenario.

Although the array division in the additive sum tree (i.e., taking the inverse of the priorities by multiplication) dramatically increases the computational complexity of vanilla PER. This may raise concerns about the feasibility of our approach. Nevertheless, this issue can be resolved by Single Instruction, Multiple Data (SIMD) structure embedded within modern CPUs. In particular, SIMD instructions can simultaneously perform the same operation, such as the array division required in our case, on all cores in parallel. Fortunately, this implementation detail is not the user's concern and can be executed implicitly by the CPU. As a result, the additional computational efficiency provided by the SIMD instructions will significantly reduce the computational burden of the LA3P framework.

6. Experiments

Here, we first briefly describe the experimental setup used to produce the reported results. Then, we compare the proposed method to the baseline methods, and conduct a set of ablation studies that further validates our theoretical conclusions.

6.1 Experimental Details

With all the mentioned concepts combined, we investigate the extent to which our prioritization framework can improve the performance of off-policy actor-critic methods. Thus, we

Algorithm 1 Actor-Critic with Loss-Adjusted Approximate Actor Prioritized Experience Replay (LA3P)

```
1: **Input:** Mini-batch size $N$, exponents $\alpha$ and $\beta$, uniform sampling fraction $\lambda$, target step-size $\zeta$, and actor and critic step-sizes $\eta_\pi$ and $\eta_Q$
2: Initialize actor $\pi_\phi$ and critic $Q_\theta$ networks, with random parameters $\phi$ and $\theta$
3: Initialize target networks $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$, if required
4: Initialize $p_{\text{init}} = 1$ and the experience replay buffer $R = \emptyset$
5: for $t = 1$ to $T$ do
6:   Select action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
7:   Store the transition tuple $\tau_t = (s_t, a_t, r_t, s_{t+1})$ in $R$ with initial priority $p_t = p_{\text{init}}$
8:   for each update step do
9:     Uniformly sample a mini-batch of transitions: $I \sim p(\tau_i) = \frac{1}{|R|}, \quad |I| = \lambda \cdot N$
10:    Optimize the critic network: $\theta \leftarrow \theta - \eta_Q \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\theta \mathcal{L}_{\text{PAL}}(\delta_\theta(\tau_i))$
11:    Compute the policy gradient $\nabla\phi(\tau_i)$ for $\tau_i$ where $i \in I$
12:    Optimize the actor network: $\phi \leftarrow \phi + \eta_\pi \cdot \frac{1}{|I|} \sum_{i \in I} \nabla\phi(\tau_i)$
13:    Update the priorities of the uniformly sampled transitions:
       $p(\tau_i) \leftarrow \max(|\delta_\theta(\tau_i)|^\alpha, 1)$ for $i \in I$
14:    Update target networks if required: $\theta' \leftarrow \zeta\theta + (1-\zeta)\theta'$, $\phi' \leftarrow \zeta\phi + (1-\zeta)\phi'$
15:    Sample a mini-batch of transitions through prioritized sampling:
       $I \sim p(\tau_i) = \frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)}, \quad |I| = (1-\lambda) \cdot N$
16:    Optimize the critic network: $\theta \leftarrow \theta - \eta_Q \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\theta \mathcal{L}_{\text{Huber}}(\delta_\theta(\tau_i))$
17:    Update the priorities of the prioritized transitions:
       $p(\tau_i) \leftarrow \max(|\delta_\theta(\tau_i)|^\alpha, 1)$ for $i \in I$
18:    Sample a mini-batch of transitions through inverse prioritized sampling:
       $I \sim \tilde{p}(\tau_i) = \frac{p_{\max}}{p(\tau_i)} = \max_i \left( \frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)} \right) \cdot \frac{\sum_j \max(|\delta_\theta(\tau_j)|^\alpha, 1)}{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}$
19:    Compute the policy gradient $\nabla\phi(\tau_i)$ for $\tau_i$ where $i \in I$
20:    Optimize the actor network: $\phi \leftarrow \phi + \eta_\pi \cdot \frac{1}{|I|} \sum_{i \in I} \nabla\phi(\tau_i)$
21:    Update target networks if required: $\theta' \leftarrow \zeta\theta + (1-\zeta)\theta'$, $\phi' \leftarrow \zeta\phi + (1-\zeta)\phi'$
22:  end for
23: end for
```

perform experiments to evaluate the effectiveness of LA3P on the standard suite of MuJoCo (Todorov et al., 2012) and Box2D (Parberry, 2013) continuous control tasks interfaced by OpenAI Gym. We evaluate the effectiveness of our method by combining it with the state-of-the-art off-policy actor-critic algorithms, namely Soft Actor-Critic (SAC) (Haarnoja et al., 2018a) and Twin Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto et al., 2018). We then compare our method against uniform sampling, PER, and the rule-based PER correction methods of LAP and MaPER. Although PAL combined with uniform sampling could also be used for comparison, Fujimoto et al. (2020) have demonstrated that it has the same expected gradient as LAP, suggesting similar empirical performance.

Our implementation of the state-of-the-art algorithms closely follows the hyperparameter setting and architecture outlined in the original papers. Particularly, we implement TD3 using the code from the author's GitHub repository[2], which contains the fine-tuned version of the algorithm. The implementation of SAC is precisely based on the original paper. Unlike the paper, we include entropy tuning, as shown by Haarnoja et al. (2018b) to improve the algorithm's overall performance. Moreover, we add 25000 exploration time steps before the training to increase the data efficiency.

We use the LAP and PAL code in the author's GitHub repository[3] to implement the algorithm and our framework, which requires a few lines on top of the standard PER implementation. Moreover, we use the same repository for the PER implementation, which is based on proportional prioritization through sum trees. To implement LA3P, we cascade uniform sampling with PAL and PER with LAP. For all experience replay sampling algorithms, except for uniform, we set $\beta = 0.4$, consistent with the original papers. We also set $\alpha = 0.6$ and $\alpha = 0.4$ for PER and LAP, respectively. Since we use LAP and PAL functions in our framework, we set $\alpha = 0.4$ for LA3P. The implementation of MaPER is obtained from the code available on the submission website[4]. Further details on the exact hyperparameter settings, architecture, and implementation can be found in Appendix B.

Each method is trained for a million steps over ten random seeds of network initialization, simulators, and dependencies, except for the Ant, HalfCheetah, Humanoid, and Swimmer environments. We found that the algorithms could benefit from further training in these environments, and thus, we train them for 2 million steps. Note that we use a replay buffer size equal to the number of training steps in all experiments. Every 1000 steps, each method is evaluated in a distinct evaluation environment (training seed + constant) for ten episodes, where no exploration and learning are performed. To construct the reported learning curves, we compute the average performance over ten evaluation episodes at each evaluation period. For easy reproducibility and fair evaluation, we did not modify the environment dynamics and reward functions of the simulators. Computing infrastructure (i.e., hardware and software) used to produce the reported results are summarized in our repository[1]. Detailed experimental setup is provided in Appendix B.

[2]`https://github.com/sfujim/TD3`
[3]`https://github.com/sfujim/LAP-PAL`
[4]`https://openreview.net/forum?id=WuEiafqdy9H`

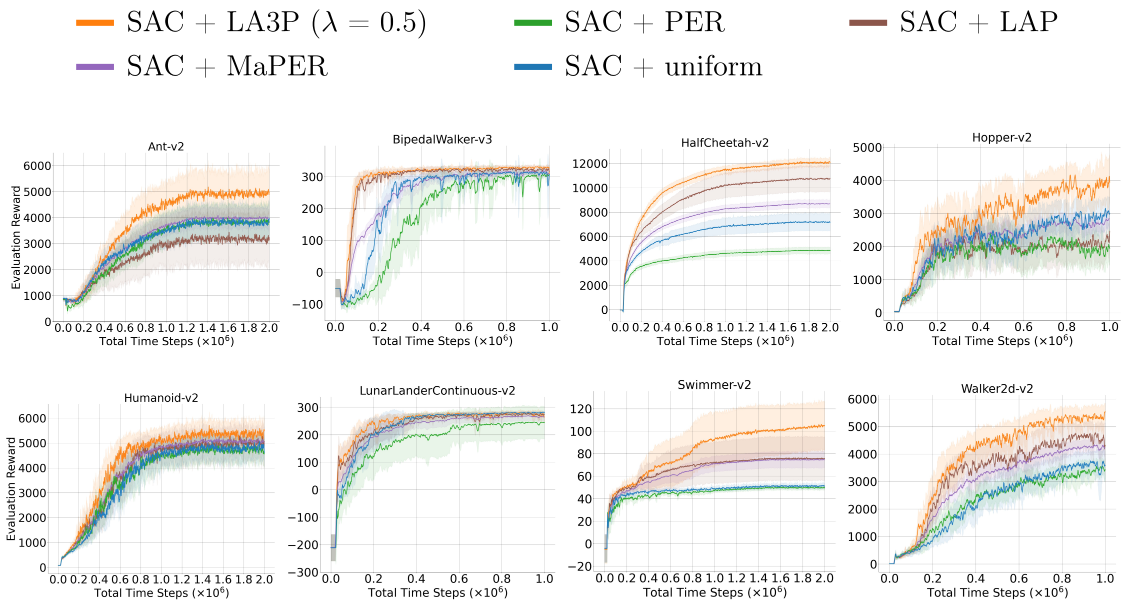

Figure 1: Learning curves for the set of MuJoCo and Box2D continuous control tasks under the SAC algorithm. The shaded region represents a 95% confidence interval over the trials. A sliding window of size 5 smoothes the curves for visual clarity.

6.2 Comparative Evaluation

Learning curves for the set of OpenAI Gym continuous control benchmarks are reported in Figures 1 and 2 for the SAC and TD3 algorithms, respectively. For all tasks, we use uniform fraction value of $\lambda = 0.5$. Initially, we tested $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ on Ant, HalfCheetah, Humanoid, and Walker2d and found that $\lambda = 0.5$ produced the best results. In the next section, we also present a sensitivity analysis based on the λ parameter. Empirical complexity analysis is provided in Appendix C.

We additionally report the average of the last ten evaluation returns in Table 1, i.e., the level where the algorithms converge. Note that for some of the tasks (e.g., HalfCheetah, Hopper, Walker2d) the baseline competing algorithms performed worse than what was reported in the original articles. This is due to the stochasticity of the simulators and used random seeds. Nonetheless, regardless of where the baselines converge, the performance disparity between the competing approaches would practically remain the same if we employed different sets of random seeds.

Based on the learning plots, we observe that LA3P performs comparably or better than the competing approaches in the majority of the tasks tested. However, in the BipedalWalker and LunarLanderContinuous environments under the SAC algorithm, LAP achieves slightly higher rewards than our method. Nonetheless, these differences are negligible, and as evidenced by the learning curves, LA3P converges faster. In these relatively trivial environments, we observe only a minor improvement. However, in the Swimmer environment, where

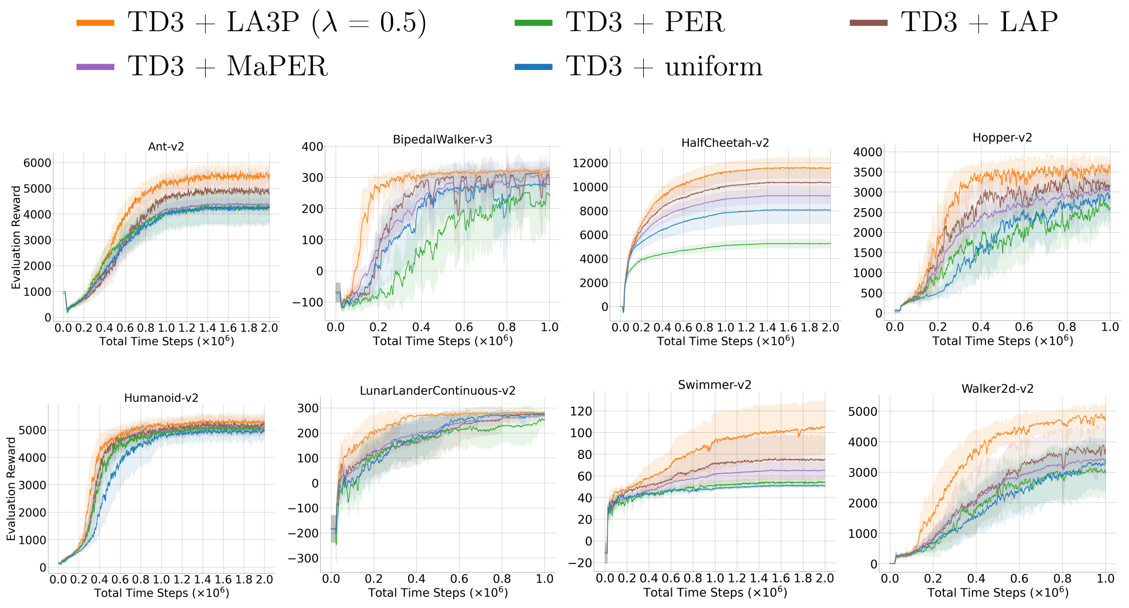

Figure 2: Learning curves for the set of MuJoCo and Box2D continuous control tasks under the TD3 algorithm. The shaded region represents a 95% confidence interval over the trials. A sliding window of size 5 smoothes the curves for visual clarity.

no algorithm could converge, the performance gains resulting from our modifications are particularly notable. Moreover, we also observe a significant improvement by LA3P in the HalfCheetah environment relative to the prior approaches. As HalfCheetah and Swimmer are considered "stable", that is, episodes terminate only after a prespecified number of time steps have been reached, these tasks entail the simulation of long horizons. Consequently, as noted by Fujimoto et al. (2020), the advantages of a corrected prioritization scheme are more prominent in environments with extended horizons.

In contrast to the learning curves, Table 1 demonstrates a high degree of overlapping confidence intervals, which may suggest that the results obtained are not significant. However, this type of hypothesis test could be misleading when there is no apparent margin between the confidence intervals. Instead, to establish whether the performance improvement provided by LA3P is substantial, we have followed the statistical testing approach described by Henderson et al. (2018) to analyze our results in depth. Henderson et al. (2018) recommend using a 2-sample t-test with a confidence interval of 0.95 as an appropriate choice to compare the performance of two algorithms. To this end, we have conducted pairwise comparisons of the last 10 evaluation rewards obtained by our algorithm and those obtained by each competing method. The resulting p-values are reported in Tables 2 and 3 for SAC and TD3, respectively. Our null hypothesis is that the difference between the last 10 evaluation returns of two algorithms is statistically insignificant. A p-value less than 0.05 indicates that there is sufficient evidence to reject the null hypothesis, leading us to conclude that the mean of the last 10 rewards for our algorithm is greater than that for the competing algorithm with

95% confidence. Conversely, if the p-value is greater than 0.05, we fail to reject the null hypothesis, and we cannot conclude that there is a significant difference the converged levels of the two algorithms.

The statistical analysis using 2-sample t-tests validates that the performance improvement offered by LA3P is statistically significant, with p-values lower than 0.05 in most of the domains tested, except for the LunarLanderContinuous and BipedalWalker environments, which are relatively trivial. It should also be noted that the difference between LA3P and LAP algorithms remains statistically insignificant in the Humanoid task under the TD3 algorithm. Nevertheless, these findings provide strong evidence that LA3P is a promising approach that can offer better performance compared to other methods. Therefore, the reward curve plots that show LA3P outperforming the competing approaches can be confidently supported by the results of the t-test per the deep RL benchmarking standards (Henderson et al., 2018), indicating the robustness and reliability of the experimental findings.

Additionally, we confirm previous empirical studies by Fujimoto et al. (2020) and Oh et al. (2022), which found that PER provides no benefits when added to off-policy actor-critic methods, and performance is usually degraded. While this is attributed to the use of MSE by Fujimoto et al. (2020), prioritization with corrected loss function (i.e., LAP) appears to have little impact in Ant, Hopper, and Walker2d compared to LA3P. In fact, the SAC algorithm is underperformed in Ant and Hopper. This result is consistent with our theoretical analysis made in Corollary 1, that is, optimizing the actor network with transitions corresponding to large TD errors can cause the approximate policy gradient to diverge from the one computed under the optimal Q-function, even if the loss function is corrected. In these environments, learning curves demonstrate that the performance gain offered by LA3P primarily comes from the inverse prioritized sampling for the actor network. This suggests that the inverse sampling in the LA3P framework plays a more significant role than the employed LAP and PAL functions. Hence, a combined solution, inverse sampling with corrected loss functions, is superior.

Finally, we notice that the performance of MaPER is not as promising as anticipated, as it only exhibits marginal enhancements over PER. We primarily attribute this unsatisfactory outcome to the model prediction structure of the algorithm. As we previously discussed, the MaPER algorithm focuses on new learnable features driven by the components in model-based RL to calculate the scores on experiences since critic networks often under- or overestimate Q-values. However, the Clipped Double Q-learning algorithm proposed by Fujimoto et al. (2018) already resolves the issues of inaccurate Q-value estimates, which is already employed in SAC and TD3. Therefore, we conclude that the main drawback of PER is not the inaccurate Q-value estimates used in the priority calculations but the biased loss function and training the actor network with large TD error transitions. Furthermore, the model prediction module in MaPER decreases the convergence rate yet, brings notable stability to the learning. Nonetheless, the resulting performance is not noteworthy. Consequently, we believe that LA3P is a preferable and comprehensive way of overcoming the underlying issues of PER in off-policy actor-critic methods.

Method	Ant	BipedalWalker	HalfCheetah	Hopper	Humanoid	LunarLanderContinuous	Swimmer	Walker2d
SAC + LA3P	**4976.28 ± 827.81**	**318.91 ± 23.13**	**12101.70 ± 315.74**	**3917.77 ± 461.58**	**5491.76 ± 189.21**	**269.06 ± 9.48**	**104.88 ± 21.33**	**5449.14 ± 265.77**
SAC + LAP	3130.02 ± 950.52	**320.05 ± 11.12**	10741.91 ± 1048.96	2324.29 ± 326.32	4927.61 ± 474.05	**272.66 ± 7.62**	75.77 ± 19.27	4403.11 ± 386.07
SAC + MaPER	3923.47 ± 369.85	**308.03 ± 13.68**	8687.17 ± 344.27	2840.91 ± 195.49	5017.23 ± 151.31	**265.22 ± 17.56**	74.64 ± 7.00	4243.83 ± 185.00
SAC + PER	3842.51 ± 628.74	**304.85 ± 5.76**	4861.91 ± 260.46	1999.58 ± 339.68	4643.74 ± 404.31	**244.62 ± 59.53**	49.94 ± 1.85	3417.08 ± 243.83
SAC	3906.49 ± 628.06	**290.44 ± 49.18**	7196.27 ± 722.02	3025.97 ± 332.20	4774.19 ± 314.03	**281.61 ± 3.26**	51.21 ± 1.74	3609.33 ± 454.04
TD3 + LA3P	**5536.67 ± 210.97**	**321.17 ± 4.11**	**11567.61 ± 833.67**	**3563.00 ± 211.70**	**5282.96 ± 224.54**	**276.60 ± 8.51**	**104.98 ± 23.62**	**4776.68 ± 339.80**
TD3 + LAP	4806.97 ± 708.24	**293.68 ± 41.43**	10343.27 ± 1081.79	3145.94 ± 416.89	**5201.14 ± 176.24**	**274.22 ± 6.72**	75.10 ± 22.19	3700.36 ± 685.96
TD3 + MaPER	4364.80 ± 209.22	**306.43 ± 24.74**	9241.38 ± 620.15	3027.40 ± 228.69	5087.84 ± 124.66	267.60 ± 7.46	65.13 ± 7.84	3410.70 ± 325.05
TD3 + PER	4269.16 ± 536.38	244.55 ± 75.49	5242.75 ± 371.66	2577.20 ± 339.48	5050.35 ± 164.63	251.96 ± 25.94	53.61 ± 3.57	3031.57 ± 826.06
TD3	4243.79 ± 582.32	**277.14 ± 74.55**	8064.88 ± 1134.70	2857.00 ± 512.74	4964.12 ± 224.71	274.95 ± 4.20	50.80 ± 1.06	3312.46 ± 822.78

Table 1: Average return of last 10 evaluations over 10 trials. Ant, HalfCheetah, Humanoid, and Swimmer are run for 2 million time steps, while the rest of the tasks for 1 million steps. ± captures a 95% confidence interval over the trials. Bold values represent the best-performing algorithm that obtains statistically significant performance improvement over the baseline under each environment.

Environment	p_{LAP}	p_{MaPER}	p_{PER}	p_{uni}
Ant	0.002	0.011	0.012	0.016
BipedalWalker	0.539	0.187	0.106	0.129
HalfCheetah	0.009	0.000	0.000	0.000
Hopper	0.000	0.000	0.000	0.001
Humanoid	0.014	0.000	0.000	0.000
LunarLanderContinuous	0.744	0.335	0.191	0.992
Swimmer	0.017	0.006	0.000	0.000
Walker2d	0.000	0.000	0.000	0.000

Table 2: The resulting p-values from a 2-sample t-test performed over the last 10 evaluation returns of LA3P and the competing methods over 10 trials under the SAC algorithm. Subscripts denote the competing method with which LA3P is compared. A p-value less than the significance level of 0.05 indicates that the difference in performance is statistically significant. Values are rounded up to three decimal points.

Environment	p_{LAP}	p_{MaPER}	p_{PER}	p_{uni}
Ant	0.024	0.000	0.000	0.000
BipedalWalker	0.084	0.107	0.024	0.107
HalfCheetah	0.029	0.000	0.000	0.000
Hopper	0.032	0.001	0.000	0.007
Humanoid	0.263	0.054	0.038	0.018
LunarLanderContinuous	0.313	0.045	0.033	0.350
Swimmer	0.026	0.002	0.000	0.000
Walker2d	0.004	0.000	0.000	0.001

Table 3: The resulting p-values from a 2-sample t-test performed over the last 10 evaluation returns of LA3P and the competing methods over 10 trials under the TD3 algorithm. Subscripts denote the competing method with which LA3P is compared. A p-value less than the significance level of 0.05 indicates that the difference in performance is statistically significant. Values are rounded up to three decimal points.

6.3 Ablation Studies

To better understand the contribution of each component in LA3P, we conduct an ablation study. The LA3P algorithm introduces several modifications to PER. In summary, LA3P consists of: (a) inverse sampling for the actor, (b) uniform sampling for the actor and critic networks to share a set of transitions, (c) the LAP function applied to the prioritized transitions, and (d) the PAL function applied to the uniformly sampled transitions.

We proceed to evaluate and discuss the performances obtained upon removing each of these components. In addition, we examine the performance of LA3P in cases where the shared transitions are low TD error experiences, instead of uniformly sampled ones, to demonstrate the reduced data efficiency previously mentioned. Moreover, we perform a sensitivity analysis for the λ parameter. We do not remove the inverse sampling as it is the backbone of our algorithm, that is, eliminating the inverse sampling for the actor network would not relate to any of the modifications introduced by this work as it would be just a mixture of uniform and prioritized sampling. To this end, we choose four challenging environments with different characteristics for a comprehensive inference. As outlined by Henderson et al. (2018) and Fujimoto et al. (2020), we consider the stable environment HalfCheetah, the unstable environment Walker2d, and the high-dimensional Ant and Humanoid environments. The latter two are widely considered to be among the most challenging environments in the MuJoCo suite (Fujimoto et al., 2020).

Table 4 presents the average of the last ten evaluation returns over ten trials for our ablation studies and sensitivity analysis, and the corresponding learning curves are depicted in Figures 3 and 4, respectively. The same experimental setup is used to perform the ablation studies, and $\lambda = 0.5$ is used for all experiments unless otherwise stated. Note that $\lambda = 0.0$ yields the LA3P setting without shared set of transitions and the evaluation results of which are already provided in Table 4 and Figure 3. In addition, $\lambda = 1.0$ corresponds to uniform sampling, which we already compare against LA3P in Figures 1 and 2, and Table 1.

First, we deduce that the set of shared transitions is the most crucial component of our framework. Independently training the actor and critic networks violate their correlation as the actor is optimized by maximizing the Q-values estimated by the Q-network, and the Q-network is trained using the actions selected by the actor. Thus, they should not be

Setting	**Ant**	**HalfCheetah**	**Humanoid**	**Walker2d**
LA3P (complete)	**5197.46 ± 162.04**	**11225.14 ± 800.59**	**5131.11 ± 193.26**	**4776.68 ± 339.80**
Low TD-error	3485.06 ± 834.26	**10992.05 ± 467.13**	3938.13 ± 1395.00	**4438.29 ± 320.47**
w/o LAP	3408.55 ± 569.04	4580.27 ± 250.82	3585.06 ± 830.28	3262.49 ± 252.69
w/o PAL	3975.29 ± 1130.62	7560.5 ± 762.22	4879.43 ± 187.23	4543.92 ± 398.97
w/o Uniform Sampling	4431.87 ± 716.48	10483.11 ± 594.31	5058.69 ± 111.42	4254.55 ± 387.41
$\lambda = 0.1$	**3768.74 ± 1007.13**	**11203.83 ± 550.87**	4695.52 ± 99.62	**4562.77 ± 372.07**
$\lambda = 0.3$	**4903.34 ± 385.48**	**10460.52 ± 933.14**	**4528.28 ± 1113.55**	**4425.08 ± 324.29**
$\lambda = 0.5$	**5197.46 ± 162.04**	**11225.14 ± 800.59**	**5131.11 ± 193.26**	**4776.68 ± 339.80**
$\lambda = 0.7$	4383.97 ± 828.71	**10656.92 ± 735.91**	4874.5 ± 130.33	**4253.76 ± 829.14**
$\lambda = 0.9$	3882.08 ± 936.85	**10547.85 ± 871.63**	3757.9 ± 1344.11	**4545.97 ± 567.28**

Table 4: Average return over the last 10 evaluations over 10 trials of 1 million time steps, comparing ablation of LA3P under low TD error shared transitions, LA3P without the LAP function, LA3P without the PAL function, LA3P without the shared set of transitions, and LA3P under $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. ± captures a 95% confidence interval over the trials. Bold values represent the best-performing configuration that obtains statistically significant performance improvement over LA3P under each environment.

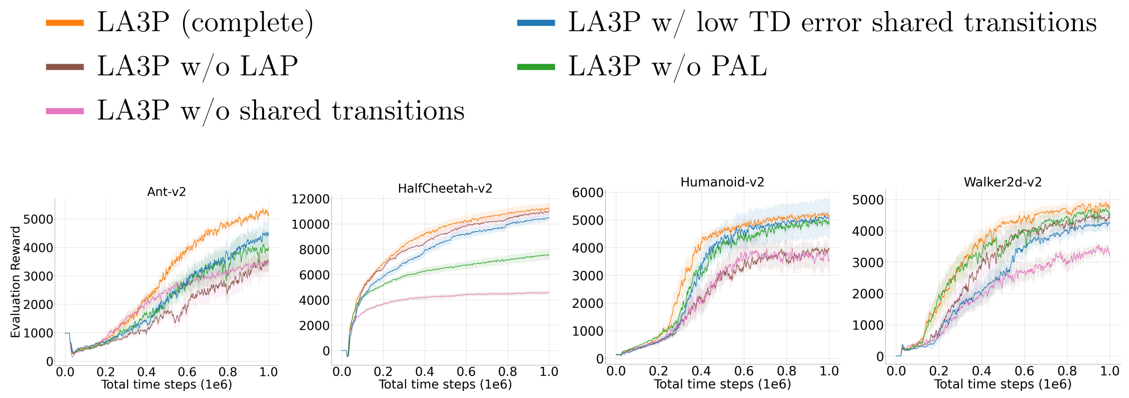

Figure 3: Learning curves for the selected MuJoCo continuous control tasks, comparing ablation of LA3P under low TD error shared transitions, LA3P without the LAP function, LA3P without the PAL function, and LA3P without the shared set of transitions. Note that the TD3 algorithm is used as the baseline off-policy actor-critic algorithm. The shaded region represents a 95% confidence interval over the trials. A sliding window of size 5 smoothes the curves for visual clarity.

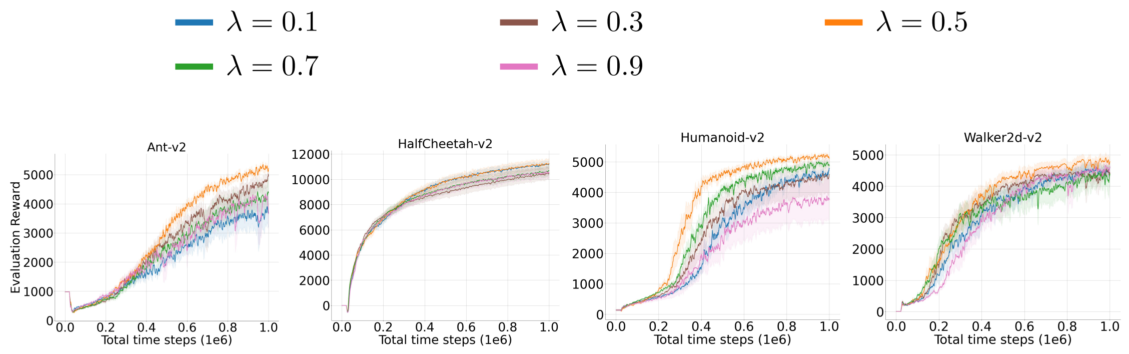

Figure 4: Learning curves for the selected MuJoCo continuous control tasks, analyzing the sensitivity of LA3P with respect to $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Note that the TD3 algorithm is used as the baseline off-policy actor-critic algorithm. The shaded region represents a 95% confidence interval over the trials. A sliding window of size 5 smoothes the curves for visual clarity.

separated in training, and we empirically verify Observation 2. Although the LAP and PAL functions apply the same number of transitions in each update step (i.e., $\lambda = 0.5$), we observe that the contribution of LAP is more significant than PAL. As discussed in our comparative evaluations, the performance improvement by LA3P largely relies on inverse sampling for the actor network. As expected, correcting the prioritization in inverse sampling for the actor network through the LAP approach, i.e., (7), is more crucial than correcting the loss by PAL for the uniformly sampled batch. Lastly, we infer that using low TD error transitions instead of uniformly sampled ones substantially degrades the performance. Although this setting would seem to be a reasonable choice at a first glance, the data efficiency notably decreases as the Q-network repeatedly trains with transitions that it has already learned well. In the expectation, the uniformly sampled batch of transitions corresponds to an intermediate TD error value compared to the transitions contained in the entire replay buffer. As we experimentally show, this may benefit both the actor and critic networks since inverse prioritized and prioritized sampling may not include transitions with intermediate TD error values, compared to the rest of the experiences.

Our sensitivity analysis on the λ parameter suggests that $\lambda = 0.5$ produces the best results across most of the environments by a considerable margin. As λ decreases, the correlation between the actor and critic networks starts to be ignored, and the performance drops. In contrast, the larger λ values yield the performance to converge to that of uniform sampling. Hence, we believe the introduced framework does not require intensive hyperparameter tuning, and $\lambda = 0.5$ can apply to many tasks.

Lastly, we conduct additional 2-sample t-tests to compare the statistical significance of each LA3P component and the selected λ value. The p-values obtained from the ablation studies and sensitivity analysis are presented in Tables 5 and 6, respectively. Our findings suggest that the reward curves obtained from the complete LA3P algorithm and the selected $\lambda = 0.5$ configuration are well-supported by statistical evidence, as confirmed by the significant contributions of each component in LA3P. Additionally, the full version of LA3P with $\lambda = 0.5$ was found to be the most effective configuration on average. Thus, the findings indicate that the complete LA3P algorithm can be a reliable and effective approach. Overall, it is shown by our ablation studies that our framework improves over the baseline actor-critic algorithms due to the structure of the introduced method rather than unintended consequences or any exhaustive hyperparameter tuning.

Environment	$p_{\text{Low-TD}}$	p_{LAP}	p_{PAL}	p_{uni}
Ant-v2	0.000	0.000	0.006	0.003
HalfCheetah-v2	0.097	0.000	0.000	0.014
Humanoid-v2	0.029	0.001	0.003	0.032
Walker2d-v2	0.059	0.000	0.164	0.017

Table 5: The resulting p-values from a 2-sample t-test performed over the last 10 evaluation returns of the complete algorithm and the ablation configuration over 10 trials. Subscripts denote the ablated component. A p-value less than the significance level of 0.05 indicates that the difference in performance is statistically significant. Values are rounded up to three decimal points.

Environment	$p_{0.1}$	$p_{0.3}$	$p_{0.7}$	$p_{0.9}$
Ant-v2	0.005	0.069	0.028	0.006
HalfCheetah-v2	0.481	0.088	0.126	0.106
Humanoid-v2	0.000	0.128	0.012	0.023
Walker2d-v2	0.175	0.054	0.106	0.221

Table 6: The resulting p-values from a 2-sample t-test performed over the last 10 evaluation returns of LA3P when $\lambda = 0.5$ and the tested λ value over 10 trials. Subscripts denote the used λ value with which $\lambda = 0.5$ is compared. A p-value less than the significance level of 0.05 indicates that the difference in performance is statistically significant. Values are rounded up to three decimal points.

7. Conclusion

In this paper, we build the theoretical foundations behind the poor empirical performance of a widely known experience replay sampling scheme, Prioritized Experience Replay (PER) (Schaul et al., 2015), in off-policy actor-critic methods. To achieve this, we first show that some transition tuples with large absolute TD errors can increase the absolute Q-value estimation error associated with the current or subsequent transition tuples. We use this finding to further indicate that training actor networks with large TD errors may cause the approximate policy gradient computed under the Q-network to diverge from the one computed under the optimal Q-function. This result emphasizes that even if the biased loss function in the PER algorithm is corrected, optimizing the actor network with low TD error transitions and Q-network with large TD error transitions can significantly increase the performance. This enables us to comprehend PER's poor performance in more detail when applied to off-policy actor-critic algorithms.

However, training actor and critic networks with different transitions throughout the learning violates the actor-critic theory since each of them is optimized with respect to each other, that is, the actor tries to maximize the Q-value estimated by the Q-network and the Q-network computes its loss based on the actions selected by the actor. This allows us to develop a novel framework, Loss Adjusted Approximate Actor Prioritized Experience Replay (LA3P), which mixes training with uniformly sampled, low, and high TD error transitions. The introduced approach also accounts for the previous findings of Fujimoto et al. (2020), which practically eliminate the outlier bias introduced by the combination of mean-squared error with PER. We evaluate the performance of LA3P using standard deep reinforcement learning benchmarks in the MuJoCo and Box2D control suite, and demonstrate that it substantially outperforms competing methods, thus improving upon the state-of-the-art. An extensive set of ablation studies further emphasizes that each LA3P component significantly impacts the offered performance improvement, and inverse prioritized sampling with corrected loss functions can increase the performance to the maximum. We firmly believe that the presented modifications, supported by comprehensive theoretical analysis, effectively address the issues with PER in off-policy actor-critic algorithms. To facilitate easy reproducibility and further research in non-uniform sampling methods, we have made the source code for our algorithm publicly available on our GitHub repository[1].

Acknowledgments

The majority of this work was completed while B. Saglam was affiliated with Bilkent University. The authors wish to thank the associate editor Prof. Alan Fern and the anonymous reviewers for their valuable and constructive feedback, which significantly enhanced the validity of the empirical results.

Appendix A. Summary of the LA3P Framework

```
uniform sampling
critic training with PAL
priority update
actor training
prioritized sampling
critic training with LAP
priority update
inverse prioritized sampling
actor training
```

Figure 5: A simplified depiction of the cascaded LA3P framework. Operations run consecutively.

Appendix B. Experimental Details

In B.1, we detail the hyperparameters and network architectures. In B.2, we describe the implementation of the algorithms used in the empirical studies. Finally, we examine the experimental setup (e.g., simulation, performance assessment) in B.3.

B.1 Architecture and Hyperparameter Setting

Architecture The off-policy actor-critic methods, TD3 and SAC, employ two Q-networks following the Clipped Double Q-learning algorithm (Fujimoto et al., 2018), and a single actor network. All networks feature two hidden layers having 256 hidden units, with ReLU activation functions after each. Following a final linear layer, the critic networks take state-action pairs (s, a) as input and output a scalar Q-value. The actor network takes state s as input and produces a multi-dimensional action a by applying a linear layer with a tanh activation function scaled with respect to the action scale of the environment.

Network Hyperparameters The Adam optimizer (Kingma & Ba, 2015) is used to train the networks, with a learning rate of 3×10^{-4} and a mini-batch size of 256. After each update step, the target networks are updated using polyak averaging with $\zeta = 0.005$, resulting in $\theta' \leftarrow 0.995 \times \theta' + 0.005 \times \theta$.

Terminal Transitions In setting the target Q-value, we use a discount factor of $\gamma = 0.99$ for non-terminal transitions and zero for terminal transitions. A transition is deemed terminal only if it stops due to a termination condition, i.e., failure or exceeding the time limit.

Actor-Critic Algorithms We use the default exploration noise $\mathcal{N}(0, \sigma_N)$ for the TD3 algorithm, as suggested by the author, where it is clipped to $[0.5, 0.5]$ with $\sigma_N = 0.2$. The range of the action space is used to scale both values. For SAC, we use the learned entropy variant, in which it is optimized to an objective of $-$`action dimensions` using an Adam optimizer with a learning rate of 3×10^{-4}. To avoid numerical instability in the logarithm operation, we cut the log standard deviation to $(20, 2)$, and add a small constant 10^{-6}, as designated by the author.

Prioritized Sampling Algorithms As described by Schaul et al. (2015), we use $\alpha = 0.6$ and $\beta = 0.4$ for PER. As LAP and PAL functions are employed in our algorithm, we directly use $\alpha = 0.4$ and $\beta = 0.4$. No hyperparameter optimization was performed on the α and β parameters since they are already fine-tuned (Fujimoto et al., 2020). Since SAC and TD3 use two Q-networks, they introduce two TD errors: $\delta_1 = y - Q_{\theta_1}$ and $\delta_2 = y - Q_{\theta_2}$, where y is the target value previously defined in (3.1). To achieve optimal performance, each priority is determined based on the maximum value of $|\delta_1|$ and $|\delta_2|$, following the approach proposed by Fujimoto et al. (2020). Similarly, newly generated samples are assigned a priority value equal to the maximum priority $p_{\text{init}} = 1$ observed during the learning process, which aligns with the structure used in PER. Lastly, applying LA3P to SAC and TD3 do not differ in terms of implementation and algorithmic setup. The main differences between these two actor-critic algorithms are the computation of the policy gradient (i.e., the use of a stochastic or deterministic policy), entropy tuning, and the presence of a target actor network. As discussed previously, Theorem 1 generalizes to both deterministic and stochastic policies. Therefore, algorithmic differences between SAC and TD3 do not regard the implementation and operation of LA3P.

Exploration To fill the experience replay buffer, the agent is not trained for the first 25000 time steps, and actions are chosen randomly with uniform probability. After that, TD3 explores the action space by introducing a Gaussian noise of $\mathcal{N}(0, \sigma_E^2 \times \texttt{max\_action\_size})$, where $\sigma_E = 0.1$ is scaled by the action space range. No exploration noise is used for SAC as it already employs a stochastic policy.

Hyperparameter Optimization No hyperparameter optimization was performed on any algorithm except for SAC. Having the remaining parameters fixed, we optimized the reward scale for the BipedalWalker, LunarLanderContinuous, and Swimmer tasks, as they were not reported in the original article. We tested the values of $\{5, 10, 20\}$, and it turned out that scaling the rewards by 5 produced the best results in these environments. In addition, all algorithms precisely adhere to the parameter settings and methodology described in their respective articles or the most recent version of their code available on their GitHub repositories. Specifically, SAC employs the identical hyperparameter configuration as outlined in the original paper, with the exception of increased exploration time steps to 25000 and entropy tuning. Regarding TD3, we used the code from the author's repository[2], which introduces minor variations in parameter settings compared to the original paper. In particular, the repository code increases the number of start steps to 25000 and batch size to 256 for all environments, which has been shown to yield better results. For LA3P, we tested $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ on the Ant, Hopper, Humanoid, and Walker2d tasks, and found that $\lambda = 0.5$ exhibited the best results. We provided the results under different λ values in our ablation studies in Section 6.3. For clarity, all hyperparameters are presented in Table 7.

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	3×10^{-4}
Mini-batch size	256
Discount factor γ	0.99
Target update rate	0.005
Initial exploration steps	25000
TD3 exploration policy σ_E	0.1
TD3 policy noise σ_N	0.2
TD3 policy noise clipping	$(-0.5, 0.5)$
SAC entropy target	`-action dimensions`
SAC log-standard deviation clipping	$(-20, 2)$
SAC log constant	10^{-6}
SAC reward scale (except Humanoid)	5
SAC reward scale (Humanoid)	20
PER priority exponent α	0.6
PER importance sampling exponent β	0.4
PER added priority constant	10^{-4}
LAP & PAL exponent α	0.4
LA3P uniform fraction λ	0.5

Table 7: The hyperparameters used in the experiments.

B.2 Implementation

TD3 is implemented using the author's GitHub repository[2], while we use the code from the same author's LAP-PAL repository[3] to implement PER and the LAP and PAL functions. The PER implementation is based on proportional prioritization through the sum tree data structure. We manually implement SAC by following the original paper and adding entropy tuning (Haarnoja et al., 2018b). Lastly, we directly use the MaPER code from the paper's submission files from the OpenReview website[4]. No changes were made to the MaPER code. Finally, the implementation of LA3P consists of the cascaded uniform, prioritized, and inverse prioritized sampling, which precisely follows the pseudocode in Algorithm 1 and the visual given in Appendix A. We do not update the priorities after the actor update with inverse prioritized sampling since the PER implementation with standard actor-critic algorithms only considers the priority updates after each critic update.

B.3 Experimental Setup

Simulation Environments All agents are assessed on continuous control benchmarks of the MuJoCo[5] and Box2D[6] physics engines, which are interfaced by OpenAI Gym[7] using v2 environments. The state-action spaces and reward functions of the environments were not modified or preprocessed for practical reproducibility and fair comparison with the empirical findings. Each environment has a multi-dimensional action space with values ranging between $[-1, 1]$, excluding Humanoid, which has a range of $[-0.4, 0.4]$.

Evaluation Every 1000 time steps, an evaluation is performed, each being the average reward over 10 episodes, using the deterministic policy from TD3 without exploration noise and the deterministic mean action from SAC. We employ a new environment with a fixed seed (training seed + constant) for each evaluation to decrease the variation caused by varying seeds. Hence, each evaluation employs the same set of initial start states.

Visualization of the Learning Curves Learning curves indicate the performance and are depicted as an average of 10 trials with a shaded region denoting a 95% confidence interval over the trials. The curves are flattened equally throughout a sliding window of 5 evaluations for visual clarity.

Statistical Testing for the Evaluation Results Consistent with the implementation outlined by Henderson et al. (2018), we employed Python's SciPy library[8] to conduct the 2-sample t-test. In the `ttest_ind` function, we specified the "greater" option for the `alternative` parameter to test whether the mean of the last 10 rewards of LA3P is superior to that of the competing algorithm. Therefore, a p-value less than 0.05 signifies sufficient evidence to reject the null hypothesis and establish with 95% confidence that the mean of the last 10 rewards of our algorithm exceeds that of the competing algorithm. Conversely, if the p-value is greater than 0.05, we cannot reject the null hypothesis, and therefore, cannot conclude that there is a substantial difference between the means of the two reward curves.

Appendix C. Empirical Complexity Analysis

Upon completion of our evaluation simulations, we conduct a comparative analysis of the runtime of PER and our algorithm, using an increased experience replay buffer size. Both sampling methods employed the off-policy actor-critic algorithms, SAC and TD3. Our analysis is performed on the Ant, HalfCheetah, Humanoid, and Swimmer environments. Specifically, we recorded the total runtime of each method when trained over 1 million and 2 million steps, using replay buffer sizes equal to the number of training steps. All experiments are conducted on a single GeForce RTX 2070 SUPER GPU and an AMD Ryzen 7 3700X

[5]`https://mujoco.org/`
[6]`https://box2d.org/`
[7]`https://www.gymlibrary.ml/`
[8]`https://scipy.org/`

8-Core Processor, which is well-suited for SIMD operations. The results of our analysis are presented in Table 8.

	Result	**PER**	**LA3P**
SAC	Runtime - 1 million steps (mins)	312.92 ± 1.63	459.18 ± 1.47
	Runtime - 2 million steps (mins)	536.46 ± 1.61	858.50 ± 1.50
	Time Increase (%)	171.44%	186.96%
TD3	Runtime - 1 million steps (mins)	145.83 ± 2.09	238.29 ± 2.13
	Runtime - 2 million steps (mins)	252.21 ± 2.15	437.18 ± 2.08
	Time Increase (%)	172.95%	183.47%

Table 8: Average runtime of PER and LA3P for 1 million and 2 million training steps under the SAC and TD3 algorithms, and the corresponding percentage increase. Values are averaged over 10 random seeds and the Ant, HalfCheetah, Humanoid, and Swimmer environments. A replay buffer size equal to the number of training steps is used in all experiments. ± captures a 95% confidence interval over the runtime.

First, our results demonstrate that the runtime of SAC is greater than that of TD3. This can be attributed to the additional entropy tuning required in SAC, which involves backpropagation and maintaining a stochastic actor. Second, our findings are consistent with the theoretical upper bound of $\mathcal{O}(\log|R|)$ for the runtime of PER, which increases logarithmically with the size of the replay buffer, rather than doubling as observed in Table 8. When we increased the size of the replay buffer from 1 million to 2 million, we also observe that the empirical runtime of LA3P increases. However, the increase is not as drastic as expected based on the theoretical runtime of $\mathcal{O}(|R|)$. It is important to note that theoretical runtime analysis provides an upper bound on the runtime, and actual empirical runtime may differ due to various practical factors that cannot be captured in the theoretical analysis. Additionally, SIMD operations could be one of the factors contributing to the observed behavior. Specifically, the computational complexity induced by LA3P is primarily due to taking the inverse of each individual element of the array by multiplication to specify which transitions to sample for training the actor network. This operation can be well-suited for SIMD operations, provided the inverse operation is well-defined for each element and there are no division-by-zero errors. In our implementation, we already use the clipping defined in (5.1) to have non-zero probability values. Therefore, the inverse operation is always well-defined for each element, and we can attribute the limitation of LA3P's runtime to the SIMD operations.

References

Andre, D., Friedman, N., & Parr, R. (1997). Generalized prioritized sweeping. In Jordan, M., Kearns, M., & Solla, S. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 10. MIT Press.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., & Lillicrap, T. (2018). Distributional policy gradients. In *International Conference on Learning Representations*.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bellman, R. E. (2003). *Dynamic Programming*. Dover Publications, Inc., USA.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *CoRR*, *abs/1606.01540*.

Fujimoto, S., Meger, D., & Precup, D. (2020). An equivalence between loss functions and non-uniform sampling in experience replay. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., & Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 33, pp. 14219–14230. Curran Associates, Inc.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In Dy, J., & Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596, Stockholmsmässan, Stockholm SWEDEN. PMLR.

Gruslys, A., Dabney, W., Azar, M. G., Piot, B., Bellemare, M., & Munos, R. (2018). The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. In *International Conference on Learning Representations*.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J., & Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018b). Soft actor-critic algorithms and applications..

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, New Orleans, Louisiana, USA. AAAI Press.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J., Leibo, J., & Gruslys, A. (2018). Deep q-learning from demonstrations. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., & Silver, D. (2018). Distributed prioritized experience replay. In *International Conference on Learning Representations*.

Isele, D., & Cosgun, A. (2018). Selective experience replay for lifelong learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

ji Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. In *Machine Learning*, pp. 293–321.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. In Solla, S., Leen, T., & Müller, K. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 12. MIT Press.

Lazaridis, A., Fachantidis, A., & Vlahavas, I. (2020). Deep reinforcement learning: A state-of-the-art walkthrough. *Journal of Artificial Intelligence Research*, *69*, 1421–1471.

Liu, R., & Zou, J. (2018). The effects of memory replay in reinforcement learning. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 478–485.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, *13*(1), 103–130.

Novati, G., & Koumoutsakos, P. (2019). Remember and forget for experience replay. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 4851–4860. PMLR.

Oh, Y., Lee, K., Shin, J., Yang, E., & Hwang, S. J. (2021). Learning to sample with local and global contexts in experience replay buffer. In *International Conference on Learning Representations*.

Oh, Y., Shin, J., Yang, E., & Hwang, S. J. (2022). Model-augmented prioritized experience replay. In *International Conference on Learning Representations*.

Parberry, I. (2013). *Introduction to Game Physics with Box2D* (1st edition). CRC Press, Inc., USA.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay.. cite arxiv:1511.05952Comment: Published at ICLR 2016.

Schlegel, M., Chung, W., Graves, D., Qian, J., & White, M. (2019). *Importance Resampling for Off-Policy Prediction*. Curran Associates Inc., Red Hook, NY, USA.

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R., Mcallester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, *12*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

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

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, *8*(3–4), 229–256.

Zha, D., Lai, K.-H., Zhou, K., & Hu, X. (2019). Experience replay optimization. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4243–4249. International Joint Conferences on Artificial Intelligence Organization.