A Proofs

Lemma 1. *If δ_θ is the temporal-difference error associated with the critic network Q_θ, then there exists a transition tuple $\tau_t = (s_t, a_t, r_t, s_{t+1})$ with $\delta_\theta(\tau_t) \neq 0$ such that the absolute temporal-difference error on τ_t is directly proportional to the absolute estimation error on at least τ_t or τ_{t+1}:*

$$|\delta_\theta(\tau_t)| \propto |Q_\theta(s_i, a_i) - Q^\pi(s_i, a_i)|; \quad i = t \vee (t+1), \tag{1}$$

where $Q^\pi(s_i, a_i)$ is the actual Q-value of the state-action pair (s_i, a_i) while following the policy π.

Proof. The proof does not consider the target networks since they aim to ensure stability and fixed objective over the updates and have no effect on the estimation [6]. First, expand $\delta_\theta(\tau_t)$ to the bootstrapped value estimation form:

$$\delta_\theta(\tau_t) = r_t + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t), \tag{2}$$

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

$$\delta^\pi(\tau_t) = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t) = 0. \tag{3}$$

Then, subtracting Equation (3) from Equation (2) yields:

$$\delta_\theta(\tau_t) = \underbrace{(Q^\pi(s_t, a_t) - Q_\theta(s_t, a_t))}_{:=x} + \gamma \underbrace{(Q_\theta(s_{t+1}, a_{t+1}) - Q^\pi(s_{t+1}, a_{t+1}))}_{:=y} \neq 0. \tag{4}$$

It is clear that x is the estimation error at time step t and y is the estimation error at the subsequent time step $t+1$. Note that the existence of an estimation error does not depend on the sign. For simplicity, we express the TD error in terms of x and y:

$$\delta_\theta(\tau_t) = x + \gamma y \neq 0. \tag{5}$$

Clearly, if $x = 0$, then $y \neq 0$, or vice versa, since $\gamma \geq 0$. Thus, there exists an estimation error by Q_θ either on τ_t or τ_{t+1} if the TD error corresponding to the Q-network is non-zero. Notice that the absolute value of the TD error in the latter equation can be directly proportional to $|x|$ or $|y|$. For instance, suppose that $\delta_\theta(\tau_t) < 0$, $x \geq 0$, and $y < 0$. In such a case, an increasing absolute TD error increases the absolute estimation error $|y|$, if x remains constant. Furthermore, one can notice that an increasing (or decreasing) absolute TD error can increase (or decrease) at least $|x|$ or $|y|$, for each combination of the signs of x, y, and $\delta_\theta(\tau_t)$. Hence, we infer that there may exist a direct proportionality between the absolute temporal-difference error $|\delta_\theta(\tau_t)|$ at time step t and the absolute estimation error for τ_t or τ_{t+1}. □

Theorem 1. *Let τ_i be a transition such that Lemma 1 is satisfied. Then, if $\delta_\theta(\tau_i) \neq 0$, the following relation holds:*

$$|\delta_\theta(\tau_i)| \propto |\nabla_\phi J(\phi(\tau_j)) - \nabla_\phi J(\phi_{\textit{true}}(\tau_j))|; \quad j = i \vee (i+1), \tag{6}$$

where $\nabla_\phi J(\phi(\tau_j))$ and $\nabla_\phi J(\phi_{\textit{true}}(\tau_j))$ are the resulting policy gradients corresponding to τ_j if computed under the Q-network Q_θ and optimal Q-function Q^π, respectively.

Proof. The proof follows from Lemma 1 and adaptation of the policy gradient theorem of Sutton et al. [8] to the deep function approximation. To begin the proof, we first formally express the standard policy iteration with function approximation in terms of the policy parameters ϕ:

$$\phi \leftarrow \phi + \eta \nabla_\phi J(\phi(s_i, a_i)), \tag{7}$$

where $\nabla_\phi J(\phi(s_t, a_i))$ is the policy gradient computed for the state-action pair $(s_i, a_i) \in \tau_i$ and η is the learning rate. Sutton et al. [8] provides a general formulation for the policy gradient in policy iteration with function approximation as:

$$\nabla_\phi J(\phi(s_i, a_i)) \coloneqq \sum_s d^\pi(s_i) \sum_a \frac{\partial \pi(s_i, a_i)}{\partial \phi} f_\theta(s_i, a_i), \tag{8}$$

where f_θ is the function approximation to Q^π, i.e., $f_\theta \coloneqq Q_\theta$, and $d^\pi(s_i)$ is a discounted weighting of encountered states starting at s_0 and then following π, denoted by:

$$d^\pi(s_i) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s_i | s_0, \pi). \tag{9}$$

Clearly, the gradient of the policy parameters ϕ is proportional to the gradient of $\pi(s_i, a_i)$ with respect to ϕ weighted by the estimated Q-value of (s_i, a_i). Here, we can neglect $d^\pi(s_i)$ since policy parameters ϕ has not effect on $d^\pi(s_i)$ [8]. Then, Equation (8) reduces to:

$$\nabla_\phi J(\phi(\tau_i)) \propto \frac{\partial \pi_\phi(s_i, a_i)}{\partial \phi} Q_\theta(s_i, a_i), \tag{10}$$

where we remind that $\nabla_\phi J(\phi(\tau_i))$ is the policy gradient computed with respect to the imperfect Q-value estimate $Q_\theta(s_i, a_i)$. Additionally, if the policy is stochastic, the latter equation transforms into:

$$\nabla_\phi J(\phi(\tau_i)) \propto \frac{\partial \log \pi_\phi(a_i | s_i)}{\partial \phi} Q_\theta(s_i, a_i). \tag{11}$$

Therefore, we generalize that the gradients of deterministic and stochastic policies are proportional to the Q-value estimates of the Q-network. Since the Q-network approximates the actual Q-function Q^π, an error exists in the value estimates of Q_θ. We express the policy gradient in terms of the true Q-value and estimation error, which is valid for both deterministic and stochastic policies:

$$\nabla_\phi J(\phi(\tau_i)) \propto (Q^\pi(s_i, a_i) + \epsilon_{\tau_i}), \tag{12}$$

where we neglect the derivatives as they do not depend on the critic and $\epsilon_{\tau_i} \in \mathbb{R}$ is the error induced by bootstrapping and function approximation in the estimation of $Q^\pi(s_i, a_i)$, i.e., $Q_\theta(s_i, a_i) = Q^\pi(s_i, a_i) + \epsilon_{\tau_i}$. Naturally, the actual Q-value of (s_i, a_i) computed under the true Q-function Q^π associates with the actual policy gradient $\nabla_\phi J(\phi_{\text{true}}(\tau_i))$:

$$\nabla_\phi J(\phi_{\text{true}}(\tau_i)) \propto Q^\pi(s_i, a_i). \tag{13}$$

From Equation (12) and Equation (13), we observe that an increasing absolute estimation error increases the divergence from the actual policy gradient since the Q-value estimate $Q_\theta(s_i, a_i)$ moves away from the actual Q-value $Q^\pi(s_i, a_i)$, which alters the weights used in policy gradient computation, i.e., Equation (10) and Equation (11). We formally express this result as:

$$|\epsilon_{\tau_i}| \propto |\nabla_\phi J(\phi(\tau_i)) - \nabla_\phi J(\phi_{\text{true}}(\tau_i))|. \tag{14}$$

According to Lemma 1, we know that a large absolute TD error can correspond to a large absolute estimation error for τ_i or τ_{i+1}. Hence, given Equation (14), the absolute estimation error in the current or subsequent step is directly proportional to the absolute TD error in the current step:

$$|\delta_\theta(\tau_i)| \propto |\epsilon_{\tau_j}|, \quad j = i \vee (i+1). \tag{15}$$

Combining Equation (14) and Equation (15), we deduce that an increasing absolute TD error in the current step increases the divergence from the actual policy gradient in the current or subsequent step:

$$|\delta_\theta(\tau_i)| \propto |\nabla_\phi J(\phi(\tau_j)) - \nabla_\phi J(\phi_{\text{true}}(\tau_j))|; \quad j = i \vee (i+1). \tag{16}$$

□

B The LA3P Framework

B.1 Pseudocode

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

Input: Mini-batch size N, exponents α and β, uniform sampling fraction λ, target learning rate ζ, and actor and critic learning rates η_π and η_Q
Initialize actor π_ϕ and critic Q_θ networks, with random parameters ϕ and θ
Initialize target networks $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$, if required
Initialize $p_{\text{init}} = 1$ and the experience replay buffer $R = \emptyset$
for $t = 1$ **to** T **do**
  Select action a_t and observe reward r_t and new state s_{t+1}
  Store the transition tuple $\tau_t = (s_t, a_t, r_t, s_{t+1})$ in R with initial priority $p_t = p_{\text{init}}$
  for each update step **do**
    Uniformly sample a mini-batch of transitions: $I \sim p(\tau_i) = \frac{1}{|R|}; \quad |I| = \lambda \cdot N$
    Optimize the critic network: $\theta \leftarrow \theta - \eta_Q \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\theta \mathcal{L}_{\text{PAL}}(\delta_\theta(\tau_i))$
    Compute the policy gradient $\nabla_\phi J(\phi(\tau_i))$ for τ_i where $i \in I$
    Optimize the actor network: $\phi \leftarrow \phi + \eta_\pi \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\phi J(\phi(\tau_i))$
    Update the priorities of the uniformly sampled transitions: $p(\tau_i) \leftarrow \max(|\delta_\theta(\tau_i)|^\alpha, 1)$ for $i \in I$
    Update target networks if required: $\theta' \leftarrow \zeta\theta + (1-\zeta)\theta'$, $\phi' \leftarrow \zeta\phi + (1-\zeta)\phi'$
    Sample a mini-batch of transitions through prioritized sampling: $I \sim p(\tau_i) = \frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_{j \in R} \max(|\delta_\theta(\tau_j)|^\alpha, 1)}; \quad |I| = (1-\lambda) \cdot N$
    Optimize the critic network: $\theta \leftarrow \theta - \eta_Q \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\theta \mathcal{L}_{\text{Huber}}(\delta_\theta(\tau_i))$
    Update the priorities of the prioritized transitions: $p(\tau_i) \leftarrow \max(|\delta_\theta(\tau_i)|^\alpha, 1)$ for $i \in I$
    Sample a mini-batch of transitions through inverse prioritized sampling: $I \sim \tilde{p}(\tau_i) = \frac{p_{\max}}{p(\tau_i)} = \max_i \left(\frac{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}{\sum_{j \in R} \max(|\delta_\theta(\tau_j)|^\alpha, 1)} \right) \cdot \frac{\sum_{j \in R} \max(|\delta_\theta(\tau_j)|^\alpha, 1)}{\max(|\delta_\theta(\tau_i)|^\alpha, 1)}$
    Compute the policy gradient $\nabla_\phi J(\phi(\tau_i))$ for τ_i where $i \in I$
    Optimize the actor network: $\phi \leftarrow \phi + \eta_\pi \cdot \frac{1}{|I|} \sum_{i \in I} \nabla_\phi J(\phi(\tau_i))$
    Update target networks if required: $\theta' \leftarrow \zeta\theta + (1-\zeta)\theta'$, $\phi' \leftarrow \zeta\phi + (1-\zeta)\phi'$
  end for
end for

B.2 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 1: A simplified depiction of the cascaded LA3P framework. Operations run consecutively.

C Theoretical Complexity Analysis

The LA3P framework introduces an additional sum tree, the LAP, and PAL functions on top of vanilla PER, which operates in $\mathcal{O}(\log|R|)$ [7]. 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, which takes $\mathcal{O}(\log|R|)$. Additionally, LA3P takes the inverse of the priorities by multiplication, which takes an additional $\mathcal{O}(|R|)$ run time. Therefore, LA3P introduces an additional $\mathcal{O}(\log|R|) + \mathcal{O}(|R|)$ complexity on top of vanilla PER. As $\mathcal{O}(|R|)$ dominates $\mathcal{O}(\log|R|)$, we conclude that LA3P has a run time 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 and may question the feasibility of our approach, it can be overcome by Single Instruction, Multiple Data (SIMD) structure supported by CPUs introduced recently. Exceptionally, SIMD instructions perform the same operation, such as the simple array division in our case, on all cores in parallel. Fortunately, this is not the user's concern and can be executed implicitly by the CPU. Therefore, we believe that the computational burden of the LA3P framework will be significantly reduced by the additional computational efficiency offered by SIMD instructions.

D Experimental Details

D.1 Architecture and Hyper-Parameter Setting

D.1.1 Architecture

The actor-critic methods, TD3 and SAC, employ two Q-networks 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 value Q. 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 multiplied by the action space scale.

D.1.2 Network Hyper-Parameters

The Adam optimizer [5] 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 in both TD3 and SAC are updated using polyak averaging with $\zeta = 0.005$, resulting in $\theta' \leftarrow 0.995 \cdot \theta' + 0.005 \cdot \theta$.

D.1.3 Terminal Transitions

In setting the target Q-value, we utilize 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.

D.1.4 Actor-Critic Algorithms

We use the default policy noise of $\mathcal{N}(0, \sigma_N)$ 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. With SAC, we utilize the learned entropy variant [3], in which entropy is optimized to an objective of $-$`action dimensions` using an Adam optimizer with a learning rate of 3×10^{-4}, similar to the actor and critic networks. To avoid numerical instability in the logarithm operation, we cut the log standard deviation to $(20, 2)$, and a small constant of 10^{-6} is added, as designated by the Haarnoja et al. [3].

D.1.5 Prioritized Sampling Algorithms

As described by Schaul et al. [7], 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 hyper-parameter optimization was performed on the α and β parameters since the used values produce the best results, as reported by Fujimoto et al. [2].

Since SAC and TD3 maintain two Q-networks, there are two TD errors defined by $\delta_1 = y - Q_{\theta_1}$ and $\delta_2 = y - Q_{\theta_2}$. Each priority considers the maximum of $|\delta_1|$ and $|\delta_2|$, as described by Fujimoto et al. [2] to produce the strongest performance. New samples are assigned a priority equal to the highest priority $p_{\text{init}} = 1$ recorded at any time during learning, as done by PER.

Applications of LA3P to SAC and TD3 do not differ in terms of implementation and algorithmic setup. The main differences between the actor-critic algorithms of SAC and TD3 are the computation of the policy gradient, entropy tuning, and the presence of the target actor network. As discussed previously, Theorem 1 is valid for deterministic and stochastic policies. Therefore, algorithmic differences between SAC and TD3 do not regard the implementation and operation of LA3P.

D.1.6 Exploration

To fill the 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 $\mathcal{N}(0, \sigma_E^2 \cdot \texttt{max action size})$, where $\sigma_E = 0.1$ is scaled by the action space range. As SAC employs a stochastic policy, no exploration noise is added.

D.1.7 Hyper-Parameter Optimization

No hyper-parameter optimization was performed on any algorithm except for SAC. Having the remaining parameters fixed, we optimized the reward scale for the BipedalWalker, LunarLander-Continuous, and Swimmer tasks, as they were not reported in the paper. We tested the values of $\{5, 10, 20\}$, and it turned out that scaling the rewards by 5 produced the best results for these environments.

All algorithms follow what is reported in the original papers or the most recent code in the respective GitHub repositories. SAC follows the precise hyper-parameter setting outlined in the original paper except for increased exploration time steps to 25000 and entropy tuning. For TD3, as we employed the code from the author's repository[1], the parameter setting has a minor difference. Different from the original paper, the code in the repository increases the number of start steps to 25000 and batch size to 256 for all environments, as reported to produce 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 6.2. For clarity, all hyper-parameters are presented in Table 1.

Table 1: Hyper-parameters used in the experiments.

Hyper-Parameter	**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

[1] `https://github.com/sfujim/TD3`

D.2 Implementation

Our implementation of the state-of-the-art algorithms closely follows the hyper-parameter setting and architecture outlined in the original papers. Particularly, we implement TD3 using the code from the author's GitHub repository[1], which contains the fine-tuned version of the algorithm. Our manual implementation of SAC is precisely based on the original paper. Unlike the paper, we include entropy tuning, as shown by Haarnoja et al. [3] to improve the algorithm's overall performance. Moreover, we added 25000 exploration time steps before the training to increase the data efficiency, as indicated by Fujimoto et al. [1]. Lastly, We directly utilize the MaPER code from the paper's submission files from the OpenReview website[2]. No changes were made to the MaPER code.

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. Specifically, LA3P is implemented by cascading uniform sampling combined with PAL and PER combined with LAP. The implementation of LA3P consists of the cascaded uniform, prioritized, and inverse prioritized sampling, which precisely follows the pseudocode in Appendix B.1. 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 update after each critic update.

D.3 Experimental Setup

D.3.1 Simulation Environments

All agents are evaluated in continuous control benchmarks of MuJoCo[4] and Box2D[5] physics engines interfaced by OpenAI Gym[6], using v2 environments. The environment, state-action spaces, and reward function are not altered or pre-processed for practical reproducibility and fair comparison with empirical findings. Each environment has a multi-dimensional action space with values ranging between $[1, 1]$, excluding Humanoid, which has a range of $[0.4, 0.4]$.

D.3.2 Evaluation

Each method is trained for a million steps over ten random seeds of network initialization, simulators, and dependencies. Every 1000 time steps, an evaluation is performed, each being the average reward over ten episodes, using the deterministic policy from TD3 without exploration noise or the deterministic mean action from SAC. We employ a new environment with a fixed seed (the training seed + a constant) for each evaluation to decrease the variation caused by varying seeds [4], so each evaluation utilizes the same set of initial start states.

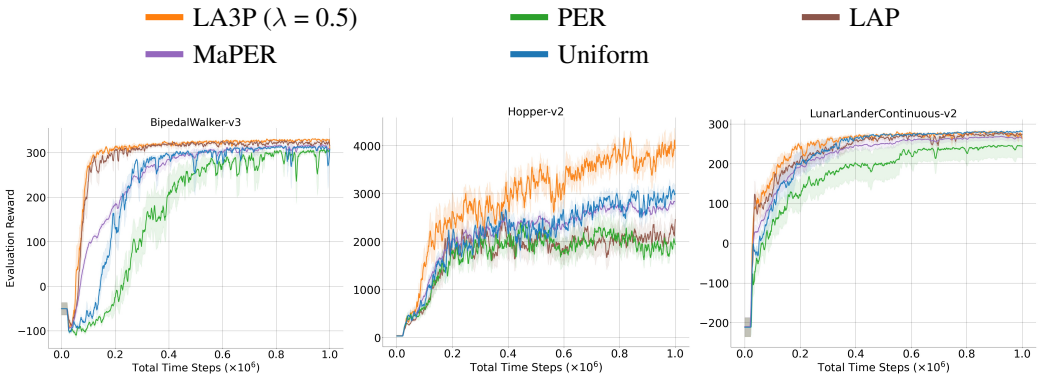
D.3.3 Visualization of the Learning Curves

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

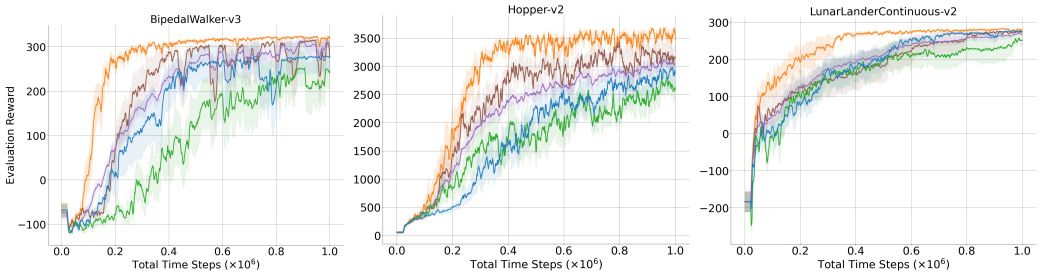
[2] https://openreview.net/forum?id=WuEiafqdy9H
[3] https://github.com/sfujim/LAP-PAL
[4] https://mujoco.org/
[5] https://box2d.org/
[6] https://www.gymlibrary.ml/

E Missing Evaluation Results

E.1 Learning Curves for Additional Environments



(a) SAC



(b) TD3

Figure 2: Learning curves for the additional set of MuJoCo and Box2D continuous control tasks under the SAC and TD3 algorithms. Curves are averaged over 10 trials, where the shaded region represents a 95% confidence interval over the trials.

E.2 Numerical Evaluation Results

Table 2: Average return of last 10 evaluations over 10 trials of 1 million time steps under the SAC algorithm. ± captures a 95% confidence interval over the trials. Bold values represent the maximum under each environment.

Environment	LA3P	LAP	MaPER	PER	Uniform
Ant	**4539.1 ± 810.2**	2934.1 ± 907.9	3675.6 ± 376.3	3605.7 ± 333.1	3519.8 ± 634.1
BipedalWalk.	318.9 ± 27.9	**320.0 ± 15.7**	308.0 ± 20.1	306.4 ± 10.5	290.4 ± 51.8
HalfCheetah	**11485.5 ± 339.9**	10223.2 ± 1016.3	8270.9 ± 326.5	6444.6 ± 897.2	6845.8 ± 686.8
Hopper	**3917.8 ± 727.5**	2324.3 ± 532.3	2840.9 ± 289.3	2420.2 ± 335.3	3026.0 ± 480.9
Humanoid	**5094.8 ± 518.2**	4726.1 ± 632.8	4821.0 ± 272.1	4652.8 ± 279.7	4567.0 ± 477.8
LunarLand.	269.1 ± 16.1	272.7 ± 11.2	265.2 ± 18.0	254.9 ± 28.5	**281.6 ± 4.6**
Swimmer	**115.8 ± 14.6**	72.0 ± 18.5	71.0 ± 6.9	59.2 ± 6.5	48.8 ± 2.2
Walker2d	**5449.1 ± 370.6**	4403.1 ± 742.6	4243.8 ± 283.9	3830.5 ± 295.5	3609.3 ± 516.2

Table 3: Average return of last 10 evaluations over 10 trials of 1 million time steps under the TD3 algorithm. $\pm$ captures a 95% confidence interval over the trials. Bold values represent the maximum under each environment.

Environment	LA3P	LAP	MaPER	PER	Uniform
Ant	**5197.5 $\pm$ 377.3**	4653.2 $\pm$ 701.2	4161.2 $\pm$ 237.3	4103.9 $\pm$ 287.9	4029.2 $\pm$ 576.3
BipedalWalk.	**321.2 $\pm$ 7.0**	293.7 $\pm$ 45.5	306.4 $\pm$ 27.9	275.5 $\pm$ 40.9	277.1 $\pm$ 74.8
HalfCheetah	**11225.1 $\pm$ 811.6**	10053.0 $\pm$ 1056.1	8975.3 $\pm$ 605.0	7035.2 $\pm$ 984.2	7824.6 $\pm$ 1091.8
Hopper	**3563.0 $\pm$ 279.3**	3145.9 $\pm$ 597.2	3027.4 $\pm$ 278.3	2802.3 $\pm$ 278.0	2857.0 $\pm$ 584.4
Humanoid	**5131.1 $\pm$ 250.8**	4998.5 $\pm$ 279.7	4938.5 $\pm$ 140.2	4916.4 $\pm$ 112.8	4802.1 $\pm$ 310.1
LunarLand.	**276.6 $\pm$ 15.1**	274.2 $\pm$ 10.4	267.6 $\pm$ 9.5	259.8 $\pm$ 13.9	275.0 $\pm$ 5.7
Swimmer	**99.3 $\pm$ 25.0**	71.3 $\pm$ 20.8	61.8 $\pm$ 7.5	56.6 $\pm$ 4.5	48.5 $\pm$ 1.3
Walker2d	**4776.7 $\pm$ 424.5**	3700.4 $\pm$ 766.2	3410.7 $\pm$ 341.8	3221.1 $\pm$ 420.9	3312.5 $\pm$ 832.2

F Empirical Complexity Analysis

Upon completing our evaluation simulations, we compare the run time of baseline uniform sampling, PER, and our algorithm. Each sampling method combines the off-policy actor-critic algorithms, SAC and TD3. We record the total run time of each method throughout our comparative evaluation experiments. All experiments are run on a single GeForce RTX 2070 SUPER GPU and an AMD Ryzen 7 3700X 8-Core Processor. Our results are presented in Table 4.

Table 4: Average run time of uniform sampling, PER, and LA3P, and their percentage increase over the off-policy actor-critic algorithms, SAC and TD3. Values are recorded over 1 million time steps and averaged over 10 random seeds and benchmark environments selected for our comparative evaluations. $\pm$ captures a 95% confidence interval over the run time.

	Result	Uniform	PER	LA3P
SAC	Run Time (mins)	225.18 $\pm$ 1.48	307.01 $\pm$ 1.52	445.42 $\pm$ 1.52
	Time Increase (%)	+0.00%	+136.34%	+197.81%
TD3	Run Time (mins)	131.51 $\pm$ 1.98	145.83 $\pm$ 2.09	238.29 $\pm$ 2.13
	Time Increase (%)	+0.00%	+110.89%	+181.19%

First, we find that the run time of SAC is greater than that of TD3. This is due to the additional entropy tuning that requires backpropagation, and maintaining a stochastic actor. Moreover, the high dimensional environments such as Ant and Humanoid significantly increase the mean run time of the algorithms. While PER has a slightly increased run time, our method considerably increases the required time for each experiment. Although this result may question the feasibility of our approach, the empirical run time is vastly lower than what is indicated by our theoretical analysis. We know that $\mathcal{O}(|R|)$ is much larger than $\mathcal{O}(\log|R|)$, mainly when the replay buffer is large. Nonetheless, this can be overcome by the discussed parallelable array division operation embedded through SIMD operations in the recently introduced CPUs.

G Learning Curves for Ablation Studies

G.1 Ablation Study of LA3P

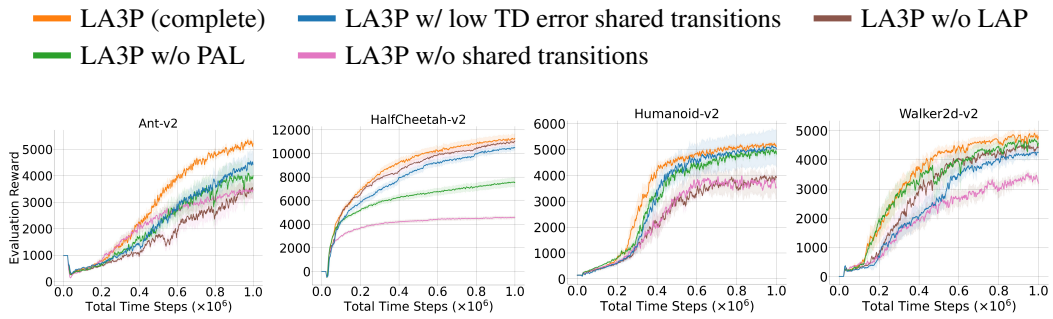


Figure 3: Learning curves for the selected MuJoCo continuous control tasks, comparing ablation of LA3P under low TD error shared transitions, LA3P without the LAP function, LA3P without the PAL function, and LA3P without the shared set of transitions. Note that the TD3 algorithm is used as the baseline off-policy actor-critic algorithm. Curves are averaged over 10 trials, where the shaded region represents a 95% confidence interval over the trials.

G.2 Sensitivity Analysis for λ

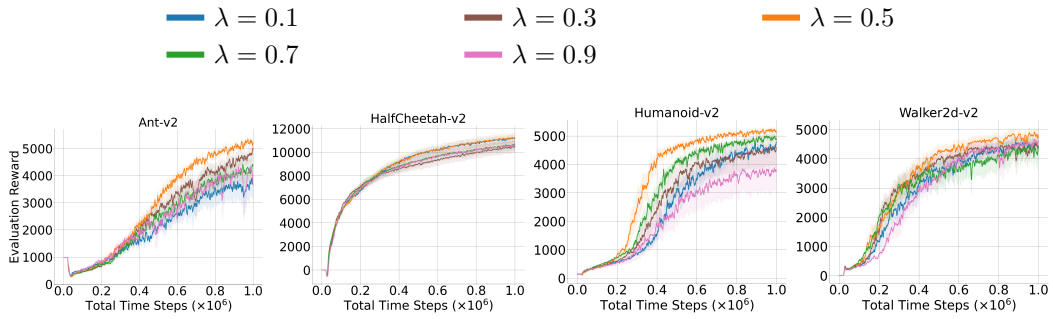


Figure 4: Learning curves for the selected MuJoCo continuous control tasks, analyzing the sensitivity of LA3P with respect to $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Note that the TD3 algorithm is used as the baseline off-policy actor-critic algorithm. Curves are averaged over 10 trials, where the shaded region represents a 95% confidence interval over the trials.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm SWEDEN, 10–15 Jul 2018. PMLR. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[2] Scott Fujimoto, David Meger, and Doina Precup. An equivalence between loss functions and non-uniform sampling in experience replay. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14219–14230. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/a3bf6e4db673b6449c2f7d13ee6ec9c0-Paper.pdf.

[3] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018. URL https://arxiv.org/abs/1812.05905.

[4] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 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, 2018. AAAI Press. ISBN 978-1-57735-800-8.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. URL http://arxiv.org/abs/1511.05952. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[8] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, 12, 02 2000.