NOVEL SAMPLING STRATEGIES FOR EXPERIENCE REPLAY MECHANISMS IN OFF-POLICY DEEP REINFORCEMENT LEARNING ALGORITHMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Furkan Burak Mutlu

Sept 2024

NOVEL SAMPLING STRATEGIES FOR EXPERIENCE REPLAY MECHANISMS IN OFF-POLICY DEEP REINFORCEMENT LEARNING ALGORITHMS

By Furkan Burak Mutlu

Sept 2024

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Sinan Gezici

Ramazan Gökberk Cinbiş

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

NOVEL SAMPLING STRATEGIES FOR EXPERIENCE REPLAY MECHANISMS IN OFF-POLICY DEEP REINFORCEMENT LEARNING ALGORITHMS

Furkan Burak Mutlu
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
Sept 2024

Experience replay enables agents to effectively utilize their past experiences repeatedly to improve learning performance. Traditional strategies, such as vanilla experience replay, involve uniformly sampling from the replay buffer, which can lead to inefficiencies as they do not account for the varying importance of different transitions. More advanced methods, like Prioritized Experience Replay (PER), attempt to address this by adjusting the sampling probability of each transition according to its perceived importance. However, constantly recalculating these probabilities for every transition in the buffer after each iteration is computationally expensive and impractical for large-scale applications. Moreover, these methods do not necessarily enhance the performance of actor-critic-based reinforcement learning algorithms, as they typically rely on predefined metrics, such as Temporal Difference (TD) error, which do not directly represent the relevance of a transition to the agent's policy. The importance of a transition can change dynamically throughout training, but existing approaches struggle to adapt to this due to computational constraints. Both vanilla sampling strategies and advanced methods like PER introduce biases toward certain transitions. Vanilla experience replay tends to favor older transitions, which may no longer be useful since they were often generated by a random policy during initialization. Meanwhile, PER is biased toward transitions with high TD errors, which primarily reflects errors in the critic network and may not correspond to improvements in the policy network, as there is no direct correlation between TD error and policy enhancement. Given these challenges, we propose a new sampling strategy designed to mitigate bias and ensure that every transition is used in updates an equal number of times. Our method, Corrected Uniform Experience Replay (CUER), leverages an efficient sum-tree structure to achieve fair sampling counts for all transitions.

We evaluate CUER on various continuous control tasks and demonstrate that it outperforms both traditional and advanced replay mechanisms when applied to state-of-the-art off-policy deep reinforcement learning algorithms like TD3 and SAC. Empirical results indicate that CUER consistently improves sample efficiency without imposing a significant computational burden, leading to faster convergence and more stable learning performance.

Keywords: experience replay, reinforcement learning, actor-critic algorithms, off-policy, deep learning, continuous control tasks.

ÖZET

DERIN DETERMINISTIK POLITIKA GRADYANI ALGORITMALARI IÇIN YENI TECRÜBE TEKRARI STRATEJILERI

Furkan Burak Mutlu
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Eylül 2024

Deneyim tekrarı, pekiştirmeli öğrenme ajanlarının geçmiş deneyimlerini tekrar tekrar kullanarak öğrenme performansını etkili bir şekilde geliştirmesini sağlar. Varsayılan deneyim tekrarı gibi geleneksel stratejiler, tekrar oynatma arabelleğinden rastgele örnekler almayı içerir, bu da farklı geçişlerin değişen önemini hesaba katmadıkları için verimsizliklere yol açabilir. Öncelikli Deneyim Tekrarı (PER) gibi daha gelişmiş yöntemler, her geçişin örnekleme olasılığını algılanan önemine göre ayarlayarak bu sorunu çözmeyi amaçlar. Ancak, her yineleme sonrasında arabellekteki her geçiş için bu olasılıkları sürekli olarak yeniden hesaplamak, hesaplama açısından maliyetlidir ve büyük ölçekli uygulamalar için pratik değildir. Ayrıca, bu yöntemler genellikle Geçici Fark (TD) hatası gibi önceden tanımlanmış metriklere dayandıkları için, aktör-kritik tabanlı pekiştirmeli öğrenme algoritmalarının performansını doğrudan artırmazlar ve bir geçişin ajanın politikasına olan önemini doğrudan temsil etmezler. Bir geçişin önemi eğitim süresince dinamik olarak değişebilir, ancak mevcut yaklaşımlar hesaplama sınırlamaları nedeniyle buna uyum sağlamakta zorlanır. Hem varsayılan örnekleme stratejileri hem de PER gibi gelişmiş yöntemler belirli geçişlere karşı yanlılıklar getirir. Varsayılan deneyim tekrarı, genellikle başlangıçta rastgele bir politika tarafından oluşturuldukları için artık faydalı olmayabilecek eski geçişleri tercih etme eğilimindedir. Öte yandan, PER, yüksek TD hatalarına sahip geçişlere karşı yanlıdır; bu da genellikle kritik ağındaki hataları yansıtır ve TD hatası ile politika geliştirme arasında doğrudan bir ilişki olmadığı için politika ağı iyileştirmelerine karşılık gelmeyebilir. Bu zorluklar göz önüne alındığında, öğrenim sırasında yanlılığı azaltmak ve her geçişin güncellemelerde eşit sayıda kullanılmasını sağlamak için yeni bir örnekleme stratejisi öneriyoruz. Yöntemimiz, Düzeltici Sabit Deneyim Tekrarı (CUER), tüm geçişler için adil örnekleme sayılarına ulaşmak

amacıyla verimli bir toplam-ağaç yapısını kullanır. CUER'in performansını çeşitli sürekli kontrol görevlerinde değerlendiriyoruz ve bunun, TD3 ve SAC gibi en gelişmiş politika dışı derin pekiştirmeli öğrenme algoritmalarına uygulandığında hem geleneksel hem de gelişmiş tekrar mekanizmalarından daha iyi performans gösterdiğini gösteriyoruz. Deneysel sonuçlar, CUER'in önemli bir hesaplama yükü getirmeden örnekleme verimliliğini sürekli olarak artırdığını, bu da daha hızlı yakınsama ve daha kararlı öğrenme performansına yol açtığını gösteriyor.

Anahtar sözcükler: deneyim tekrarı, pekiştirmeli öğrenme, aktör-kritic algoritmaları, derin öğrenme, sürekli kontrol görevleri.

Acknowledgement

I would like to extend my heartfelt thanks to all the people who have supported me throughout my master's journey. Above all, I am profoundly thankful to my mother, father, and sister for their unwavering love, constant encouragement, and belief in my abilities. Their generosity, combined with their continuous support, has been the source of my strength, determination, and motivation to persevere, even in the most challenging moments.

I am deeply grateful to my advisor, Prof. Süleyman Serdar Kozat, for his unwavering trust in me and for giving me a valuable place in his research team. His mentorship has not only guided me in becoming a better researcher but also enriched my perspective in ways that will benefit me throughout my life. Beyond academic guidance, he shared insights that helped me develop a balanced outlook and refine my approach to both professional endeavors and personal challenges. His encouragement and thoughtful guidance have profoundly shaped my understanding of the world, and I will always be indebted to him for the invaluable support and direction he has provided during my academic journey.

I would also like to extend my heartfelt thanks to my incredible friends who stood by my side throughout this journey, always offering their unwavering support. Their constant encouragement helped me stay motivated and focused, even during the most challenging moments. I am deeply grateful for their presence and the positivity they brought into every step of this process. Last but not least, I owe a special thank you to my girlfriend, who continuously cheered me on and supported me with endless patience and understanding. Her unwavering belief in me made this journey far more manageable, and she has been a constant source of happiness, making me a better and happier person through her love and encouragement.

I thank to Vodafone for their support with 5G and Beyond Graduate Scholarship Programme.

Contents

1 Introduction **1**

1.1 Preliminaries 1

1.2 Contributions 2

1.3 Outline 3

2 Background **4**

2.1 Reinforcement Learning 4

2.1.1 Off-Policy Learning 9

2.2 Twin Delayed DDPG 12

2.3 Soft Actor Critic 15

2.4 Experience Replay Technique 16

2.4.1 Prioritized Experience Replay (PER) 19

2.4.2 Combined Experience Replay (CER) 21

3 Prioritizing Batches via Corrected Uniform Experience Replay (CUER) 23

3.1 Corrected Uniform Experience Replay 24

3.1.1 Motivation 25

3.1.2 Batch Producing Strategy 29

3.2 Experiments 32

3.2.1 Simulation Environments 32

3.2.2 Implementation Details 35

3.2.3 Results for TD3 Algorithm 36

3.2.4 Results for SAC Algorithm 41

3.2.5 Ablation Studies 45

4 Conclusion 50

List of Figures

2.1 Agent-Environment Interaction in Reinforcement Learning 6

3.1 Sampling counts of transitions during the training with a uniform sampling strategy. 25

3.2 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Ant-v4 environment 38

3.3 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in HalfCheetah-v4 environment 38

3.4 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Hopper-v4 environment 39

3.5 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Humanoid-v4 environment 39

3.6 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in LunarLanderContinuous-v2 environment 40

3.7 Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Walker2d-v4 environment 40

3.8 Learning progress plot of SAC for vanilla experience replay, PER and CUER in Ant-v4 environment 42

3.9 Learning progress plot of SAC for vanilla experience replay, PER and CUER in HalfCheetah-v4 environment 42

3.10 Learning progress plot of SAC for vanilla experience replay, PER and CUER in Hopper-v4 environment 43

3.11 Learning progress plot of SAC for vanilla experience replay, PER and CUER in Humanoid-v4 environment 43

3.12 Learning progress plot of SAC for vanilla experience replay, PER and CUER in LunarLanderContinuous-v2 environment 44

3.13 Learning progress plot of SAC for vanilla experience replay, PER and CUER in Walker2d-v4 environment 44

3.14 Sampling counts of transitions during the training with CUER. . . 45

3.15 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Ant-v4 environment 47

3.16 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in HalfCheetah-v4 environment 47

3.17 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Hopper-v4 environment 48

3.18 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Humanoid-v4 environment 48

3.19 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in LunarLanderContinuous-v2 environment 49

3.20 Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Walker2d-v4 environment 49

List of Publications

1. Mutlu, Furkan. B., Sarp Yenicesu, A., Kozat, S. S., and Oguz, O. S., “CUER: Corrected Uniform Experience Replay for Off-Policy Continuous Deep Reinforcement Learning Algorithms”, arXiv e-prints, 2024. doi:10.48550/arXiv.2406.09030.

2. Baturay Saglam, Furkan B. Mutlu, Dogan C. Cicek, and Suleyman S. Kozat. "Actor Prioritized Experience Replay." J. Artif. Int. Res. 78 (Jan 2024). https://doi.org/10.1613/jair.1.14819

3. D. C. Cicek, E. Duran, B. Saglam, F. B. Mutlu, and S. S. Kozat, ‘Off-Policy Correction for Deep Deterministic Policy Gradient Algorithms via Batch Prioritized Experience Replay’, in 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), 2021, pp. 1255–1262.

4. B. Saglam, F. B. Mutlu, and S. S. Kozat, ‘An Optimistic Approach to the Temporal Difference Error in Off-Policy Actor-Critic Algorithms’, in 2022 IEEE Symposium Series on Computational Intelligence (SSCI), 2022, pp. 875–883.

Chapter 1

Introduction

1.1 Preliminaries

Deep reinforcement learning (DRL) has emerged as a powerful approach for solving complex decision-making tasks across various domains, such as video games [1], continuous control problems [2], board games [3], and real-time strategy games [4]. DRL algorithms leverage function approximation techniques, like neural networks, to derive policies that are near-optimal without needing to explore every possible state-action pair exhaustively. This capability has allowed DRL agents to achieve remarkable success in environments where traditional methods struggle to generalize or scale effectively [5].

However, a critical challenge in DRL is managing the correlation in the data presented to the learning algorithms [6, 7]. As agents collect experiences based on their current policies, the data becomes temporally correlated, which violates the assumption of independent and identically distributed (i.i.d.) samples required by many optimization techniques, such as stochastic gradient descent. This correlation can significantly hinder the stability and convergence of learning processes [8, 9]. Experience replay, a widely adopted solution to this issue, mitigates the problem by storing past experiences in a buffer and sampling them

uniformly to create batches of decorrelated data. This process enhances the sample efficiency and improves the stability of learning by ensuring the experiences resemble an i.i.d. distribution [6, 10, 11].

Despite its advantages, uniform sampling introduces its own set of biases. Over time, older transitions have a higher chance of being sampled simply due to their longer presence in the buffer, which can skew the agent's learning towards outdated experiences. This situation creates a shift in the distribution of sampled experiences, moving the learning process further off-policy. Off-policy learning occurs when the agent learns from data collected under different policies than its current one, potentially leading to what is known as "soft divergence," where the agent's performance deteriorates as training progresses due to outdated or irrelevant data [12].

To address these challenges, this thesis introduces the Corrected Uniform Experience Replay (CUER) method, which aims to balance the sampling probabilities of transitions across the entire history, thereby reducing biases and making the sampling distribution closer to a uniform one. CUER is designed to provide a fair representation of experiences, thereby improving the overall performance and convergence speed of off-policy reinforcement learning algorithms.

1.2 Contributions

The contributions of this thesis are summarized as follows:

1. **Novel Experience Replay Method:** We propose Corrected Uniform Experience Replay (CUER), a new experience replay strategy that aims for fairness by ensuring a more balanced sampling probability across the entire transition history. This method dynamically adjusts the sampling distribution, reducing the overemphasis on older transitions that is prevalent in traditional experience replay strategies.

2. **Computational Efficiency:** CUER offers an efficient and straightforward implementation that can substitute any existing experience replay method. By eliminating the need for recalculating priorities for every transition after each iteration, CUER reduces computational overhead, making it suitable for large-scale reinforcement learning tasks.

3. **Empirical Validation:** Extensive experiments were conducted using various environments from the OpenAI Gym and MuJoCo suites. CUER was compared against state-of-the-art experience replay methods, demonstrating significant improvements in performance, sample efficiency, and convergence speed across almost all tested environments.

4. **Broader Applicability:** While our method specifically targets off-policy learning scenarios, CUER's design is versatile enough to be adapted to other learning paradigms, offering potential benefits in a wide range of reinforcement learning applications.

By introducing CUER, this thesis contributes a more balanced and computationally efficient experience replay strategy that mitigates the biases inherent in existing methods, thereby enhancing the stability and effectiveness of deep reinforcement learning algorithms.

1.3 Outline

The structure of this thesis is organized as follows: Chapter 2 introduces the background material, relevant literature, and specific algorithms that are combined with our proposed Corrected Uniform Experience Replay (CUER) method. Chapter 3 represents the CUER methodology, the theoretical principles that underpin it, the experimental framework, implementation details, empirical findings and ablation studies. Chapter 4 concludes the thesis by summarizing the main findings, contributions, and proposes potential directions for future work in improving new experience replay strategies in reinforcement learning.

Chapter 2

Background

In this chapter, we provide a comprehensive overview of the reinforcement learning framework, focusing on its application to continuous control tasks. We summarize two state-of-the-art off-policy deep reinforcement learning methods designed for such tasks and delve into the mechanisms that underpin experience replay [10]. Specifically, we examine two major methodologies for experience replay: Prioritized Experience Replay (PER) and Combined Experience Replay (CER) [13, 14]. These approaches have been developed to optimize the learning process by improving the agent's utilization of past experiences.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make sequential decisions by interacting with a dynamic environment. The agent aims to maximize cumulative rewards, which are scalar signals provided by the environment in response to its actions. RL is often modeled as a Markov Decision Process (MDP), a mathematical framework used to describe decision-making problems where outcomes are partly random and partly under the control of a decision-maker.

At each discrete time step t, the agent observes a state $s_t \in S$ from a set of possible states and selects an action $a_t \in A$ according to a policy $\pi(a|s_t)$, which can be either deterministic or stochastic. After executing the action a_t, the agent receives a scalar reward r_t and transitions to a new state s'_t as determined by the probabilistic transition dynamics of the environment$P(s', r|s, a)$. This sequence- observing a state, selecting an action, receiving a reward, and transitioning to a new state- constitutes the fundamental reinforcement learning cycle.

A core concept in modern reinforcement learning is the use of experience replay [10]. Experience replay involves storing the agent's interactions with the environment in a replay buffer-a memory repository that holds past experiences, typically represented as tuples (s, a, r, s'). During training, the agent samples from this replay buffer to update its policy or value function, rather than relying solely on the most recent experience. This approach offers several advantages:

- **Breaking Temporal Correlations**: By randomly sampling experiences from the buffer, the agent reduces the correlation between consecutive experiences, which is a common issue when learning online.
- **Efficient Use of Data**: It allows the agent to reuse past experiences multiple times, which improves sample efficiency, particularly in environments where data collection is expensive or slow.
- **Improved Stability**: Experience replay helps stabilize the training of deep neural networks by smoothing out the updates and avoiding abrupt changes that could result from correlated experiences.

The primary objective of a reinforcement learning agent is to maximize its cumulative expected return. The return G_t from a given state is defined as the discounted sum of future rewards:

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \tag{2.1}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards relative to immediate rewards. A higher value of γ places more emphasis

on long-term rewards, while a lower value focuses on short-term gains. The choice of γ plays a crucial role in shaping the agent's behavior, balancing the trade-off between short-term and long-term objectives.

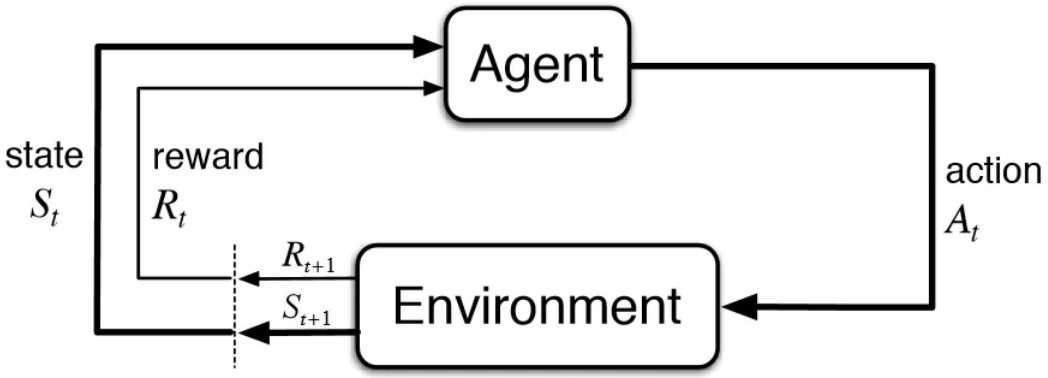

Figure 2.1: Agent-Environment Interaction in Reinforcement Learning

In addition to the fundamental reinforcement learning model, the agent's policy π evolves through continuous optimization, informed by feedback from the environment in the form of scalar rewards. Figure 2.1 illustrates the key components of a reinforcement learning framework, showing how the agent interacts with the environment to maximize its cumulative reward over time.

Reinforcement learning algorithms can be broadly categorized into on-policy and off-policy methods. In on-policy methods, the policy being optimized is the same as the policy used to make decisions, while off-policy methods allow for separate behavior and target policies. This flexibility is particularly advantageous in continuous control tasks, such as robotic manipulation or automated driving, where exploration can be costly or dangerous [6].

Two prominent off-policy algorithms that have shown state-of-the-art performance in continuous control tasks are Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) [15, 16]. These algorithms utilize experience replay to train neural networks that approximate optimal policies in high-dimensional action spaces, each offering unique strategies to address specific challenges in reinforcement learning.

Twin Delayed Deep Deterministic Policy Gradient (TD3): TD3 builds upon the Deep Deterministic Policy Gradient (DDPG) algorithm, addressing key limitations like overestimation bias and instability. TD3 introduces several improvements over DDPG:

- **Two Critic Networks**: TD3 uses two critic networks to estimate the value of actions, taking the minimum value predicted by the critics to reduce overestimation bias.
- **Delayed Policy Updates**: The policy network is updated less frequently than the value networks, which helps stabilize learning.
- **Target Policy Smoothing**: TD3 adds noise to the target policy's actions, providing smoother target values and reducing variance, which enhances learning stability.

Soft Actor-Critic (SAC): SAC is an off-policy algorithm that combines the benefits of both maximum entropy reinforcement learning and actor-critic methods. It aims to maximize both the expected cumulative reward and the entropy of the policy, encouraging exploration and preventing premature convergence to suboptimal policies. Key features of SAC include:

- **Entropy Regularization**: SAC optimizes a trade-off between maximizing the expected reward and the entropy of the policy. This results in more diverse actions, promoting better exploration and reducing the risk of getting stuck in local optima.
- **Stochastic Policy**: Unlike deterministic algorithms like DDPG and TD3, SAC uses a stochastic policy that models the probability distribution over actions. This approach allows the agent to effectively explore high-dimensional action spaces.
- **Efficient Sample Utilization**: SAC employs experience replay and learns from off-policy data, improving sample efficiency and enabling effective learning from diverse past experiences.

The efficiency of learning and the quality of the policy are heavily influenced by the strategy for managing the replay buffer and the specifics of the algorithmic approach employed. Advanced techniques such as Prioritized Experience Replay (PER) and Combined Experience Replay (CER) further refine this process by intelligently selecting which experiences to replay based on their potential for learning gain, thereby accelerating the learning process and improving the robustness of the policy developed.

Prioritized Experience Replay (PER): Unlike uniform sampling, where all experiences are equally likely to be chosen, PER samples experiences based on their temporal-difference (TD) error. The intuition is that experiences with higher TD errors are more unexpected and, therefore, more valuable for learning. PER assigns a probability of being sampled to each experience based on its TD error, ensuring that the agent focuses more on experiences that have a greater potential to improve the policy.

Example: In a robotics application, an experience where a robot arm narrowly misses an object may have a high TD error due to its unexpectedness. Sampling this experience more frequently can help the agent learn to make more precise movements.

Combined Experience Replay (CER): CER integrates elements of both uniform and prioritized replay strategies. While it leverages the benefits of prioritized sampling, it also retains a fraction of experiences sampled uniformly to maintain diversity and prevent overfitting to a narrow set of experiences. This approach strikes a balance between focusing on high-impact experiences and preserving the breadth of learning from diverse interactions.

By integrating these elements, reinforcement learning provides a powerful framework for teaching agents to perform complex tasks in uncertain and dynamic environments. The development of efficient learning algorithms and effective management of experience replay are key areas of ongoing research and innovation in this field.

2.1.1 Off-Policy Learning

In off-policy reinforcement learning, agents gather transitions experienced during training as tuples (s, a, r, s') and store them in an experience replay buffer [10]. This replay buffer allows agents to reuse past experiences multiple times to refine their policies and value networks, thereby enhancing data and sampling efficiency.

Off-policy methods differ from on-policy methods in that they allow the learning of a target policy π that can differ from the behavior policy used to generate the data. This flexibility enables off-policy algorithms to learn from experiences generated by different policies, including those collected by suboptimal or exploratory behaviors. This ability to decouple the policy being learned from the policy used to gather data is particularly advantageous in situations where collecting new data is expensive, time-consuming, or risky.

For example, in applications such as robotics, autonomous driving, or financial trading, direct experimentation in the real world is costly or dangerous. On-policy methods, which require continuous interaction with the environment to update the policy, can be impractical in these cases because they often need a large number of interactions to converge. In contrast, off-policy methods can leverage pre-collected datasets or simulated experiences, allowing the agent to learn effectively without the need for constant, real-time interaction.

Moreover, off-policy methods provide greater sample efficiency by reusing and replaying past experiences stored in a replay buffer. This reuse reduces the dependency on fresh data from the environment, thereby speeding up the learning process. Off-policy algorithms, such as Q-learning, Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC), can learn from any previous experience, regardless of the data's source, which is critical for training in environments where generating new data is either limited or constrained by physical or financial realities.

Additionally, off-policy methods allow for a more exploratory behavior while

maintaining learning stability. They can experiment with diverse strategies during data collection while still training toward a more refined target policy. This ability to explore more widely is crucial in complex environments where optimal policies are unknown beforehand and require extensive exploration to discover [17].

Off-policy methods evaluate value functions not just based on the states visited but also the action choices of the policy, leading to the definition of the action-value function or Q-function Q^π [18]. The Q-function, often referred to as the critic, predicts the expected return R from following the policy π after taking action a in state s:

$$Q^\pi(s,a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a \right]. \tag{2.2}$$

where E_π denotes the expectation under the policy π, and $\gamma in [0,1]$ is the discount factor that determines the importance of future rewards.

The Q-function can also be computed recursively using the Bellman equation [19]:

$$Q^\pi(s,a) = E_{r,s' \sim P_\pi, a' \sim \pi} \left[r + \gamma Q^\pi(s',a') \right], \tag{2.3}$$

where P_π represents the environment dynamics under policy π, a' is the action taken according to the policy π in the next state s', and r is the immediate reward received after taking the action a in state s.

In deep off-policy reinforcement learning, the Q-function (critic) is typically estimated by deep neural networks parameterized by θ, denoted as Q_θ. Off-policy Temporal Difference (TD) learning is employed by minimizing a loss function based on the Temporal Difference error, which measures the discrepancy between the predicted Q-value and the target Q-value [20]. The target Q-value is computed as:

$$y(\tau) = r + \gamma Q_{\theta'}(s',a'), \tag{2.4}$$

where $\tau = (s,a,r,s')$ represents a transition and $Q_{\theta'}$ is an independent target network with parameters θ'. Then, off-policy Temporal Difference (TD) learning

is conducted by minimizing a loss $\delta_\theta(\tau)$ based on Temporal Difference error:

$$\delta_\theta(\tau) = y(\tau) - Q_\theta(\tau). \tag{2.5}$$

The target network $Q_{\theta'}$ plays a crucial role in stabilizing the learning process by providing a fixed target for the TD error minimization [21]. It operates with a delayed set of parameters θ', which are periodically updated from the primary network parameters θ. This mechanism, known as a *target network*, helps reduce the risk of divergence by decoupling the moving target in the update equations, thereby stabilizing the learning process.

In off-policy algorithms, such as Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Soft Actor-Critic (SAC), an actor-critic architecture is employed. Here, the critic estimates the Q-function, while the actor represents the policy network π_ϕ parameterized by ϕ. The actor network aims to maximize the Q-value estimated by the critic, leading to the following optimization objective:

$$\phi \leftarrow \phi + \eta \nabla_\phi Q_\theta(s, \pi_\phi(s)), \tag{2.6}$$

where η is the learning rate for the actor update.

The action a' in the target equation (2.4) can be determined either by the primary actor network π_ϕ, which defines the policy for choosing actions, or by the target actor network $\pi_{\phi'}$, depending on the specific method employed. The target actor network, parameterized by ϕ', functions similarly tı the target Q-network, providing consistent target values to stabilize training.

The separation between the primary and target networks is crucial because it reduces the moving targets in the update equations, leading to more stable updates. The TD error expressed as:

$$\delta_\theta(\tau) = y(\tau) - Q_\theta(\tau). \tag{2.7}$$

remains central to evaluating the performance of the agent. While both formulations of the TD error are valid ($\delta_\theta(\tau) = y(\tau) - Q_\theta(\tau)$ and $\delta_\theta(\tau) = Q_\theta(\tau) - y(\tau)$),

the choice of notation is used to emphasize different aspects of the learning process. Here, we employ a distinct notation for the TD error since it will be regularly referenced in subsequent discussions.

In contrast, on-policy methods, such as Proximal Policy Optimization (PPO) [22] or Trust Region Policy Optimization (TRPO) [23], update the policy based solely on the most recent experience trajectory, which can be less sample efficient and often requires more interactions with the environment. Off-policy methods, by leveraging stored experiences and separate behavior policies, provide greater flexibility and are generally more suitable for environments where exploration is costly or where sample efficiency is paramount.

By effectively utilizing experience replay and optimizing both the actor and critic networks with stable targets, off-policy learning offers a powerful approach for training agents in complex, continuous control tasks, allowing for more efficient learning and robust performance.

2.2 Twin Delayed DDPG

One of the groundbreaking off-policy deep reinforcement learning methods that successfully addresses the challenge of high-dimensional continuous control tasks is Twin Delayed Deep Deterministic Policy Gradients (TD3) [15]. While the Deep Deterministic Policy Gradient (DDPG) algorithm was the first continuous control methodology introduced in off-policy deep reinforcement learning literature, it suffers from overestimation bias, which can hinder convergence to optimal policies across many tasks [2, 15].

TD3 was developed to enhance the stability and accuracy of policy learning in continuous control tasks by addressing specific shortcomings of its predecessor, DDPG. One of the primary enhancements introduced by TD3 is its approach to mitigating the overestimation bias inherent in value-based reinforcement learning algorithms [15]. Overestimation bias arises when the estimated Q-value of a

state-action pair (s, a), denoted by $\hat{Q}^{\pi}(s, a)$, exceeds its true Q-value $Q^{\pi}(s, a)$:

$$\hat{Q}^{\pi}(s, a) > Q^{\pi}(s, a). \tag{2.8}$$

Overestimation bias tends to propagate through the training updates, where overestimated Q-values are repeatedly used to update other values. This leads to a positive feedback loop that exacerbates the bias, potentially causing instability in learning and even leading to catastrophic forgetting, where the agent suddenly loses previously acquired knowledge.

TD3 introduces several innovations to counteract this bias and improve learning performance:

Clipped Double Q-Learning

TD3 uses an approach known as Clipped Double Q-Learning to reduce overestimation. Unlike DDPG, which relies on a single Q-network, TD3 utilizes two critic networks Q_1 and Q_2, which independently estimate the Q-values of the same state-action pair. The target Q-value is computed using the minimum of the two estimates:

$$y = r + \gamma \min_{i=1,2} Q'_i(s', a'), \tag{2.9}$$

where $Q'_i(s', a')$ represents the target Q-value produced by the i-th critic network, r is the immediate reward and γ is the discount factor, and $a' = \pi'(s') + \epsilon$ is the action taken by the policy network π' with added noise $\epsilon \sim \mathcal{N}(0, \sigma)$.

By using the minimum value between the two critics, TD3 effectively provides a more conservative estimate of the target Q-value, thereby mitigating the risk of overestimation. This technique is particularly effective because it biases the learning process toward underestimating rather than overestimating Q-values, which is less likely to lead to instability in learning.

Delayed Policy Updates

A crucial innovation in TD3 is the use of delayed policy updates. In DDPG, both the actor (policy) and critic (value) networks are updated simultaneously at each time step. However, this approach can lead to instability as the actor network is trained on noisy, frequently changing value estimates.

To address this, TD3 updates the actor network less frequently than the critic networks, typically every two or more critic updates. Formally, if τ represents a time step, the policy is updated only at time steps satisfying:

$$\tau, \quad \text{mod } d = 0, \tag{2.10}$$

where d is the delay parameter, typically set to 2 or more. This delay in policy updates allows the critic networks to provide more accurate estimates of the Q-values, resulting in more reliable gradients for updating the policy network. This mechanism helps prevent the actor network from overfitting to temporary inaccuracies in the Q-function, thus enhancing the overall stability of learning.

Target Policy Smoothing

Another key feature of TD3 is target policy smoothing, designed to address the deterministic exploitation of Q-function inaccuracies. When calculating the target Q-value, TD3 adds noise to the action produced by the target policy network, which helps smooth the Q-value estimates:

$$a' = \pi'(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c), \tag{2.11}$$

where $\mathcal{N}(0, \sigma)$ is Gaussian noise with mean zero and standard deviation σ, and c is a small constant that limits the noise range. This noise addition makes the policy less sensitive to slight variations in action values, reducing the likelihood of the policy overfitting to the peculiarities or inaccuracies of the Q-function. It also ensures more robust performance in continuous control tasks, where fine adjustments to actions can significantly impact outcomes.

By combining Clipped Double Q-Learning, delayed policy updates, and target policy smoothing, TD3 addresses the critical limitations of DDPG, leading to more stable and reliable policy learning. These innovations allow TD3 to achieve state-of-the-art performance in various high-dimensional continuous control tasks, making it a powerful algorithm for real-world applications where robustness and stability are paramount.

The integration of two critic networks reduces the mean squared error (MSE) of Q-value predictions by effectively halving the variance of estimation errors,

assuming that the errors of the two critics are independent. Moreover, the delayed policy update strategy allows the critic networks to evolve with greater precision, providing smoother and more accurate gradients for the actor network.

Overall, TD3 represents a significant advancement over DDPG by addressing its weaknesses through carefully designed modifications that promote stable and efficient learning, even in complex, high-dimensional environments.

2.3 Soft Actor Critic

Soft Actor-Critic (SAC) is an advanced off-policy algorithm in deep reinforcement learning that addresses the critical challenges of sample efficiency and stability, particularly in high-dimensional continuous action spaces. Derived from the maximum entropy reinforcement learning framework [24–26], SAC not only aims to maximize the expected reward but also encourages the policy to explore by maximizing entropy. This dual objective leads to robust and explorative behavior, enhancing performance in complex environments [16].

The SAC algorithm utilizes an actor-critic structure with two key components. Actor is the policy network π_ϕ, which outputs a probability distribution over actions. It is typically parameterized as a Gaussian distribution, and the network predicts the mean and the covariance of the action distribution given a state. SAC also employs two Q-function estimators (Critics), Q_{θ_1} and Q_{θ_2}, to reduce positive bias in the estimation of the expected reward, a known issue in Q-learning algorithms which can lead to overestimation.

The objective function for the actor is designed to maximize both the expected return and the entropy of the policy, providing a natural exploration incentive. The entropy term $\log \pi_\phi(a_t|s_t)$ acts as a regularizer, promoting policy diversity. The actor's objective can be expressed as follows:

$$J(\pi) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi} \left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t) \right], \tag{2.12}$$

where Q_θ represents the minimum of the two Q-function estimates.

The twin Q-functions are updated using the Bellman equation with an entropy-augmented reward:

$$Q_{\text{target}} = r(s_t, a_t) + \gamma \left(\min_{i=1,2} Q_{\theta_i}(s_{t+1}, \tilde{a}_{t+1}) - \alpha \log \pi_\phi(\tilde{a}_{t+1}|s_{t+1}) \right), \tag{2.13}$$

where $\tilde{a}_{t+1}$ are sampled actions from the current policy, and α is the temperature parameter that balances the trade-off between maximizing entropy and reward [27].

A distinctive feature of SAC is its dynamic adjustment of the temperature parameter α, which is crucial for controlling the stochasticity of the policy. The objective for adjusting α aims to maintain a predefined target entropy level $\bar{H}$, leading to the following optimization problem:

$$\min_{\alpha} \mathbb{E}_{a_t \sim \pi_\phi}[-\alpha \log \pi_\phi(a_t|s_t) - \alpha \bar{H}], \tag{2.14}$$

This adaptive mechanism ensures that the entropy of the policy remains close to $\bar{H}$, which is particularly useful in diverse and unpredictable environments.

Empirical evaluations demonstrate that SAC outperforms traditional algorithms like DDPG and TD3, particularly in tasks with complex, high-dimensional action spaces. The introduction of entropy maximization significantly improves both the efficiency and robustness of policy learning, leading to faster convergence and more stable performance across different tasks and settings.

2.4 Experience Replay Technique

Experience Replay (ER) is a pivotal technique in training deep reinforcement learning algorithms that deal with the correlated nature of sequential data gathered from interactions with the environment. In reinforcement learning, successive state transitions are inherently correlated since the state at time $t+1$ (denoting as s') is usually the same as the state at time t (denoted as s), with only minor changes in state values. This is particularly true in real-world scenarios where state transitions tend to be smooth, and deviations from s to s' often contain

small alterations, resulting in correlated transition tuples. Such correlations can adversely affect the learning process, as neural networks generally perform better with uncorrelated samples [10, 28].

To mitigate the challenges posed by correlated samples, the experience replay mechanism employs a cyclic replay buffer. This buffer stores the experiences—defined as transitions consisting of state, action, reward, and subsequent state—gathered by an agent while interacting with the environment. The stored transitions are then used to update the agent's policy and value functions. This approach helps to decorrelate the training samples by enabling the agent to learn from a randomized subset of past experiences rather than solely from consecutive ones. In its most basic form, the experience replay technique samples transitions uniformly from the buffer, implying that each transition has an equal probability of being selected for replay. Under uniform sampling, the probability of sampling a specific transition $\tau = (s, a, r, s')$ at any training time T is given by:

$$P(\tau) = \frac{1}{N(T)}, \tag{2.15}$$

where $N(T)$ represents the total number of transitions in the buffer at time T. This sampling strategy introduces a temporal bias: transitions collected early in the training process, which reside in the buffer longer, are more likely to be sampled multiple times compared to those added later. This bias can potentially skew the learning process towards earlier experiences, affecting the overall training dynamics and the development of the agent's policy. This problem and potential mitigations will be investigated in detail in further sections.

While uniform sampling treats all transitions with equal importance, various methods have been developed to prioritize transitions based on their potential contribution to the learning process. Prioritized Experience Replay (PER) [13] adjusts the sampling probability of each transition based on the magnitude of its temporal difference (TD) error, which serves as an indicator of the unexpectedness of an outcome:

$$\delta = r + \gamma Q'(s', a'; \theta') - Q(s, a; \theta), \tag{2.16}$$

where θ and θ' are the parameters of the primary value network and the target value network, respectively. This approach prioritizes transitions that are likely to

have a larger impact on the learning process, thereby accelerating the convergence of the algorithm.

Building on the concept of prioritizing valuable experiences, Focused Experience Replay (FER) enhances the strategy by specifically increasing the sampling probabilities of more recent transitions, acknowledging their greater relevance to the current state of the learning model [29]. In addition to FER, Hindsight Experience Replay (HER) [11] introduces a novel perspective by allowing the agent to learn from alternate outcomes that were not the original goals of the episode. In environments with sparse rewards, HER redefines a failed experience as a successful one by shifting the goal post-episode to a state that was actually achieved. This significantly enhances learning from limited feedback by leveraging every possible experience as a learning opportunity. Lastly, Combined Experience Replay (CER) [14] modifies the traditional experience replay by always including the most recent transition in the training batch. This method addresses the challenge of large replay buffers where older data might overshadow more relevant recent experiences. By integrating the latest experiences into each training batch, CER ensures that the learning process remains dynamically updated, enhancing the model's responsiveness and reducing temporal bias [14]. Furthermore, the experience replay process be optimized as a learning task [30]. For instance, Experience Replay Optimization (ERO) introduces a new policy called replay policy, which inherently learns the probabilities of transitions in the buffer to feed the agent with the most useful transitions during the training [30]. Neural Experience Replay Sampler takes the idea of employing an additional replay policy one step further and introduces a novel neural network architecture to estimate the usefulness of a transition by considering many local and global features such as state transition tuple, TD error, timestamp information, and reward improvement over evaluation periods [31].

Experience replay methodology significantly enhances the efficiency and effectiveness of deep reinforcement learning by reducing the detrimental effects of correlated training samples. Through innovations such as prioritized and dynamic sampling, this technique continues to evolve, offering robust solutions to the complex challenges faced in training reinforcement learning agents.

2.4.1 Prioritized Experience Replay (PER)

Prioritized Experience Replay (PER) is an advanced technique within the domain of deep reinforcement learning that addresses the limitations of traditional experience replay methods. In standard experience replay, all transitions are treated equally during the training phase, regardless of their significance to the agent's learning process [13]. This uniform sampling can be suboptimal because it may lead to inefficient learning, where less informative transitions are replayed just as frequently as those that could provide valuable learning signals. PER improves upon this by adjusting the replay probability of each transition based on its importance, typically using the temporal difference (TD) error as a proxy for significance.

The central idea behind PER is that certain experiences are more valuable to the agent's current learning objectives and should, therefore, be replayed more often. The measure of an experience's value is often determined by the magnitude of the TD error, which quantifies how surprising or informative the transition is for the agent. Transitions with a high TD error suggest a significant difference between the predicted Q-value and the observed reward, indicating a learning opportunity. The TD error for a transition is defined as:

$$\delta = r(s, a, s') + \gamma Q'(s', a'; \theta') - Q(s, a; \theta), \tag{2.17}$$

where $r(s, a, s')$ is the reward received after taking the action a in state s and moving to the state s', γ is the discount factor, Q and Q' are the value network and target value network, respectively, and θ, θ' are their corresponding parameters. By prioritizing experiences based on the absolute TD error, PER enables the agent to focus on transitions that provide the greatest learning potential, thus accelerating the learning process and improving sample efficiency.

In PER, each transition stored in the replay buffer is assigned a priority level that is updated as the agent learns. The priority level p_i of the i-th transition is typically set proportional to the absolute TD error:

$$p_i = |\delta_i| + \epsilon, \tag{2.18}$$

where δ_i is the TD error of the i-th transition, and ϵ is a small positive constant added to ensure all transitions have a non-zero probability of being replayed. This prioritization encourages the replay of experiences that most significantly deviate from the current value estimates, thereby providing stronger learning signals.

While PER focuses on high-priority transitions, it is essential to maintain a balance between exploring diverse experiences and exploiting the most informative ones. To achieve this, PER incorporates a stochastic sampling method that probabilistically selects transitions based on their priority. The probability of sampling transition i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{2.19}$$

where p_i is the priority of the i-th transition and α is a hyperparameter that controls the degree of prioritization, with $\alpha = 0$ corresponding to uniform random sampling and higher values giving more weight to high-priority transitions.

A primary challenge with PER is the bias introduced by non-uniform sampling, which changes the expected distribution of updates. To correct for this bias, PER uses importance sampling (IS) weights to adjust the magnitude of each sampled transition's update. The IS weight for transition i is calculated as:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta, \tag{2.20}$$

where N is the size of the replay buffer, and β controls how much these weights affect the learning updates, with $\beta = 0$ meaning no correction and $\beta = 1$ providing full correction. The IS weights are normalized so that their maximum value is 1, ensuring that no single transition excessively dominates the learning update. During training, the loss function for each sampled transition is scaled by its corresponding weigh w_i to compensate for the biased sampling, improving the convergence properties of the learning algorithm.

Empirical evidence shows that Prioritized Experience Replay (PER) can lead to quicker convergence and enhanced performance across various tasks, particularly in complex environments where some experiences are significantly more

informative than others. This efficiency gain is most apparent in tasks characterized by sparse rewards. In these scenarios, traditional methods may struggle to effectively identify and leverage the infrequent but valuable rewarding experiences. However, PER typically exhibits less effectiveness in environments with continuous action spaces, especially in the context of actor-critic methods. A key reason for this is that PER prioritizes transitions based on the temporal-difference (TD) error calculated solely from the critic network. This error indicates the surprise or unexpectedness of a transition according to the current value estimates of the critic, but it does not necessarily imply usefulness or relevance for improving the actor network. Since the actor network is responsible for selecting actions, a high TD error in the critic does not guarantee that adjusting the actor in response to this transition will lead to better policy performance. This disconnection can lead to inefficient learning, where the actor does not gain as much from the experiences deemed important by the critic, resulting in suboptimal policy updates and convergence challenges in continuous action domains. These issues will be further investigated in later chapters of this thesis, where a method to mitigate this problem will also be proposed.

2.4.2 Combined Experience Replay (CER)

Combined Experience Replay (CER) is a variant of experience replay that aims to improve learning speed and stability by incorporating both recent and past experiences into the agent's training process. Unlike other replay methods that might prioritize transitions solely based on their temporal-difference (TD) error or uniformly sample experiences, CER ensures that the most recent transition is always included in the training batch. This approach addresses some of the issues inherent in traditional experience replay, particularly when dealing with large replay buffers [14]. The fundamental idea behind CER is to combine online learning with experience replay to make the training process more robust. In CER, each training batch is composed of both the most recent transition and a sampled minibatch from the replay buffer. By always including the latest transition, CER ensures that the agent immediately learns from its most recent

experience, which is particularly useful when dealing with large replay buffers where a new transition might otherwise not be used for learning until much later.

A critical problem with traditional experience replay is that as the size of the replay buffer increases, the probability that a newly added transition will be replayed soon decreases. For a replay buffer of size m, if we sample one transition from the replay buffer per time step, the probability that a new transition t will be replayed within the next k time steps (where $k \leq m$) is given by:

$$P(\text{replay within } k) = 1 - \left(1 - \frac{1}{m}\right)^k. \tag{2.21}$$

This function is monotonically decreasing with increasing buffer size m. Thus, with a larger replay buffer, a rare transition is likely to influence learning only much later. If this transition happens to be crucial, the delay in replaying it can slow down the overall learning speed, as the agent misses out on learning from a significant experience in a timely manner.

CER directly addresses this problem by ensuring that all transitions, especially the most recent ones, influence the agent immediately. By always including the most recent transition in the training batch, CER reduces the sensitivity of learning performance to the size of the replay buffer. This immediate incorporation of new experiences helps the agent adjust its policy more rapidly in response to recent changes in the environment, thus enhancing the speed and stability of learning.

The combination of the latest transition with a minibatch sampled from the replay buffer can be expressed as:

$$\text{Training Batch} = \{(s_t, a_t, r_t, s_{t+1})\} \cup \text{Sampled Minibatch}. \tag{2.22}$$

Here, (s_t, a_t, r_t, s_{t+1}) represents the most recent transition, while the sampled minibatch contains transitions from various points in the past.

Chapter 3

Prioritizing Batches via Corrected Uniform Experience Replay (CUER)

In this chapter, we introduce a novel experience replay method, Corrected Uniform Experience Replay (CUER), designed to improve the sample efficiency and stability of off-policy continuous deep reinforcement learning algorithms [32]. We begin by discussing the limitations of existing experience replay prioritization methods, particularly the inefficiencies introduced by reassessing the importance of every transition after each sampling step. This inefficiency often leads to excessive computational overhead, reducing the overall performance of reinforcement learning algorithms.

Furthermore, we explore the impact of deviations from the agent's most recent policy due to outdated transitions stored in the replay buffer. Such deviations increase the frequency of off-policy updates, negatively affecting the agent's performance by causing more significant divergence from the optimal policy. CUER addresses these issues by employing a corrected uniform sampling strategy that balances the sampling probability among all stored experiences while dynamically adapting to the changing importance of transitions as the agent's policy evolves.

We then present the key mechanisms of CUER, including its ability to maintain fairness in sampling across the entire transition history and its effectiveness in minimizing off-policyness by stochastically sampling experiences to more closely match the distribution of on-policy data. Additionally, we provide a detailed analysis of how CUER adjusts the state distribution to remain more aligned with the current policy, thereby enhancing the overall training stability and convergence rate of the learning algorithm.

Lastly, we elaborate on the implementation details of CUER, its compatibility with existing reinforcement learning frameworks, and the empirical results demonstrating significant improvements in performance and convergence speed across various continuous control tasks in OpenAI Gym and MuJoCo environments. We also discuss how CUER's simple yet effective approach to experience replay can be easily integrated into current reinforcement learning methodologies, offering a viable alternative to conventional sampling techniques.

3.1 Corrected Uniform Experience Replay

In this section, we outline the motivation behind Corrected Uniform Experience Replay (CUER) and the specific challenges it aims to address in off-policy deep reinforcement learning. We provide a detailed explanation of the batch-producing strategy used in CUER, which balances the need for fairness among all stored experiences while dynamically adjusting to changes in transition importance. Additionally, we discuss the issues related to computational inefficiency and off-policy updates that CUER seeks to mitigate, ensuring improved sample efficiency and stability during the training process.

3.1.1 Motivation

Experience replay has become a fundamental component in deep reinforcement learning, allowing agents to efficiently leverage past experiences for better learning. However, traditional methods such as uniform sampling and Prioritized Experience Replay (PER) each have limitations that can negatively impact learning performance.

Uniform Sampling treats every stored transition equally, regardless of its relevance to the agent's current learning objectives. While this method ensures a broad exploration of the state-action space and maintains decorrelation between sampled transitions—helping the data resemble an independently and identically distributed (i.i.d.) scenario—uniform sampling introduces an unintended bias. Older transitions that have been in the replay buffer longer are more likely to be sampled, skewing the state distribution and making the learning process more "off-policy." This off-policy effect occurs when the agent learns from experiences collected under policies that differ from its current one, potentially leading to a phenomenon known as "soft divergence," where the agent's performance degrades over time due to an accumulation of outdated experiences.

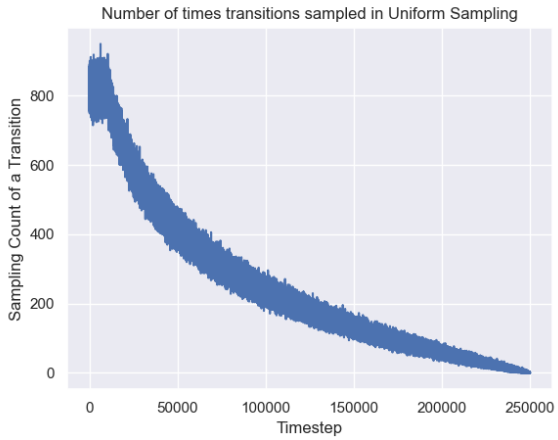

Figure 3.1: Sampling counts of transitions during the training with a uniform sampling strategy.

Figure 3.1 demonstrates that older transitions are sampled more frequently than newer ones, resulting in an increased off-policy effect. This occurs because transitions that have remained in the replay buffer longer have a higher probability of being selected, which can cause the agent to learn from outdated experiences that are less aligned with its current policy."

On the other hand, Prioritized Experience Replay (PER) aims to mitigate the limitations of uniform sampling by assigning higher sampling probabilities to transitions deemed more significant for learning, typically based on their temporal-difference (TD) error. However, this prioritization process is not without drawbacks. Constantly recalculating the importance of each transition is computationally intensive, especially in large-scale applications where the replay buffer contains many transitions. Moreover, PER can lead to overemphasis on certain transitions, resulting in excessive updates to both the actor and critic networks, which can inadvertently push the learning process further off-policy. The degree of these adverse effects is often directly related to the intensity of the prioritization.

Additionally, the significance of any given transition changes dynamically throughout training as the agent's policy and value function are iteratively refined. This dynamic nature makes it impractical to continuously reassess the priority of every transition stored in the replay buffer. The expectation of interval between consecutive sampling of a particular transition, i, can be expressed as:

$$T_i = \frac{1}{P_i N} \tag{3.1}$$

where P_i denotes the probability of selecting the i-th transition at a given timestep, and N represents the batch size. This method assumes that the importance of a transition remains unchanged until it is sampled again. However, a transition initially considered valuable may lose its relevance or even become detrimental as the agent's learning progresses — and the opposite can also occur. Given these dynamics, an algorithm that relies heavily on prioritizing transitions could potentially lead to a less effective learning process. In some cases, the basic Experience Replay (ER) strategy, which randomly samples transitions without

prioritization, may outperform methods that selectively prioritize experiences in the replay buffer [14]. The PER approach typically recalculates the sampling probability of a transition only when it is sampled, assuming its importance remains constant until the next sampling. This assumption can lead to suboptimal learning, as a transition initially considered beneficial may become irrelevant or even detrimental as training progresses, and vice versa.

Given these challenges, the Corrected Uniform Experience Replay (CUER) method addresses these shortcomings by introducing a stochastic sampling technique that maintains fairness across all stored experiences while dynamically adapting to changes in the importance of transitions.

The Corrected Uniform Experience Replay (CUER) method introduces several key contributions to the field of deep reinforcement learning, particularly in the context of off-policy learning algorithms. One of the primary advantages of CUER is its ability to ensure fairness in sampling. Unlike traditional methods, which either prioritize certain transitions excessively (as in PER) or favor older data (as in uniform sampling), CUER maintains a balanced sampling probability distribution across the entire transition history. This balanced approach prevents the biases inherent in both uniform and prioritized sampling, ensuring that all stored experiences have a fair chance of being sampled, leading to a more effective learning process.

Moreover, CUER significantly enhances computational efficiency by stochastically adjusting sampling probabilities without requiring constant recalculations. This efficiency is particularly valuable for large-scale reinforcement learning tasks where computational resources are often a limiting factor. By avoiding the intensive computation typically associated with recalculating priorities, CUER offers a more practical solution that reduces the overall computational burden, making it suitable for real-time applications and environments with large replay buffers.

CUER also promotes improved on-policy learning by aligning the state distribution of sampled experiences more closely with the agent's current policy. This is achieved through a corrected uniform sampling strategy that takes into account

the dynamic nature of transition importance. As a result, CUER reduces the frequency of off-policy updates, which can otherwise negatively impact learning by creating a mismatch between the experiences the agent is learning from and the policy it is currently using. This alignment helps ensure that the learning process remains more stable and consistent, particularly in dynamic environments where the policy may change frequently.

Additionally, CUER enhances sample efficiency and training stability by balancing the sampling of both recent and historically important transitions, creating a more stable training environment that accelerates convergence and improves overall performance in continuous control tasks. This approach leverages a wide range of experiences while maintaining alignment with the current policy, leading to a more robust learning process. Unlike other strategies like PER, which can reduce performance in actor-critic settings, CUER consistently boosts performance by avoiding excessive prioritization and maintaining better policy consistency.

In reinforcement learning, the "deadly triad" refers to the combination of function approximation, bootstrapping, and off-policy learning. When these three elements are present, algorithms can suffer from unbounded value estimates, which negatively affect the learning progress of the agent [6]. Among these elements, function approximation is essential, especially in complex environments with vast state and action spaces, where it is impractical to explore every possible state-action pair. While Monte Carlo learning offers an alternative by avoiding bootstrapping, it relies on complete trajectories that reach a terminal state, making it unsuitable for tasks without well-defined endpoints.

Opting for on-policy learning over off-policy learning eliminates the risks associated with learning from experiences gathered under different policies, but it also prevents the agent from utilizing valuable past experiences. This can result in highly correlated transitions that disrupt the training of neural networks [6]. To mitigate the last aspect of the deadly triad, off-policy learning, adjusting the sampling probabilities of transitions can help improve algorithm performance on various tasks [6].

Given these challenges, there is a need for a refined sampling strategy that balances the representation of valuable transitions without overly skewing the learning trajectory. In this context, we propose Corrected Uniform Experience Replay (CUER), a novel experience replay prioritization method designed to minimize off-policy updates in off-policy deep reinforcement learning algorithms. CUER strategically samples experiences to ensure fairness across all stored transitions and dynamically adapts to their changing importance, thereby aligning the sampled state distribution more closely with the current policy and improving overall learning stability.

3.1.2 Batch Producing Strategy

The Corrected Uniform Experience Replay (CUER) method employs a dynamic and adaptive strategy for producing training batches, designed to maximize learning efficiency while minimizing off-policy effects. This strategy addresses the inherent limitations of both uniform sampling and prioritized experience replay (PER) by effectively balancing the sampling probabilities across all transitions stored in the replay buffer.

When a new transition, t_i, is added to the replay buffer, CUER assigns it an initial high priority to ensure it is frequently sampled in the early stages after its addition. This approach guarantees that the newest experiences, which are often the most relevant to the agent's current policy, are promptly integrated into the learning process. The initial sampling probability, $P(t_i)$, for new transition is set to the maximum priority in the buffer:

$$P(t_i) = \frac{B}{\Upsilon}, \tag{3.2}$$

where Υ denotes the total sum of priorities for all transitions currently stored in the buffer, and B is the batch size. This high initial priority allows the agent to quickly learn from new data, promoting rapid adaptation to recent changes in the environment. Algorithm 1 presents Corrected Uniform Experience Replay (CUER).

Algorithm 1 Corrected Uniform Experience Replay (CUER)

Require: Experience replay buffer $\mathcal{B}$ of size M, batch size B, total training steps T, sum tree structure for sampling

Initialize replay buffer $\mathcal{B} \leftarrow \{\}$
for each transition t_i added to $\mathcal{B}$ **do**
    Set initial priority $P(t_i) \leftarrow \max(P) + \epsilon$
end for
for each training step $t = 1, \ldots, T$ **do**
    Sample a minibatch $\mathcal{S} \subset \mathcal{B}$ of size B using sum tree with probability distribution given in Eq. 3.4
    for each sampled transition $t_i \in \mathcal{S}$ **do**
        Update priority $P(t_i)$ (Eq. 3.3) ▷ Decrease priority to reduce future sampling probability
    end for
    Normalize priorities to maintain $\sum_{i=1}^{M} \Pr(t_i) = 1$
end for
Output: Updated priorities and experience replay buffer $\mathcal{B}$

At each training step, CUER samples transitions from the replay buffer using a sum tree structure, which efficiently manages the sampling and updating processes based on the assigned priorities. Each time a transition t_i is sampled, its priority is dynamically decreased to gradually reduce its sampling probability over time. This update mechanism is designed to ensure that no single transition dominates the sampling process, thus maintaining a fair distribution of learning opportunities across all experiences. The updated priority, $P'(t_i)$, of a sampled transition is computed as:

$$P'(t_i) = \frac{P(t_i) * \Upsilon - 1}{\Upsilon} \tag{3.3}$$

where $P(t_i)$ is the current priority of a transition, and Υ is the sum of all sample priorities in the buffer. This decrement prevents the overrepresentation of any single transition, such as those collected early in the training process, which might otherwise dominate the learning updates and bias the agent's policy away from recent and potentially more relevant experiences, thereby promoting a more uniform exploration of the experience buffer.

CUER continuously adjusts the sampling probabilities of all transitions in the buffer to reflect their updated priorities after each training iteration. The idea behind this correction is to eliminate any bias introduced by the prioritized sampling similar to that used in PER [13]. The probability of sampling a specific transition t_i at a given step is recalculated to ensure that the probability distribution adapts to the dynamic nature of the learning process:

$$\Pr(t_i) = \frac{P(t_i)}{\sum_{j=1}^{N} P(t_j)}, \tag{3.4}$$

where $\Pr(t_i)$ is the normalized probability of sampling transition t_i, $P(t_i)$ is the priority of t_i, and N is the total number of transitions in the replay buffer. This normalization ensures that the sum of all sampling probabilities equals 1, thereby maintaining a proper probability distribution across all stored experiences.

To maintain an equitable sampling strategy, CUER is designed such that the expected number of times a transition is sampled during the training process is balanced across all experiences. The expected number of times a transition t_i is sampled, denoted as $\mathbb{E}[\text{Count}_{t_i}]$, can be expressed as:

$$\mathbb{E}[\text{Count}_{t_i}] = \frac{B \cdot T}{M}, \tag{3.5}$$

where B is the batch size, N is the total number of training steps, and M is the size of the experience replay buffer. This expression indicates that, on average, each transition is sampled in proportion to the batch size, the total number of training steps, and the size of the experience replay buffer.

3.2 Experiments

This section presents a comprehensive evaluation of the Corrected Uniform Experience Replay (CUER) algorithm through a series of experiments conducted in various simulation environments. We first describe the learning environments used to assess the performance of CUER, detailing the specific tasks and challenges each environment presents. Following this, we outline the implementation details of the experiments, including the chosen hyperparameters and other training configuration settings. Finally, we share the empirical results, comparing CUER against other state-of-the-art reinforcement learning methods to demonstrate its effectiveness and superiority in enhancing learning stability, sample efficiency, and overall performance.

3.2.1 Simulation Environments

The simulation environments chosen for evaluating the CUER algorithm include a carefully selected set of benchmarks from the OpenAI Gymnasium and MuJoCo platforms [33,34]. These environments are widely recognized in the reinforcement learning community for their diverse range of tasks and complexities, providing a robust foundation for assessing advanced learning algorithms under varying conditions. Each baseline reinforcement learning algorithm is also evaluated within these same environments to ensure a fair and comprehensive comparison of performance, stability, and sample efficiency across all methods.

3.2.1.1 Ant-v4

The Ant-v4 environment from the MuJoCo suite simulates a 3D robotic ant with a torso and four legs, each controlled by applying torques at eight hinge joints. The goal is to learn effective movement in the forward direction. The environment's action space is continuous, allowing torques to vary between -1.0 and 1.0, while the observation space includes 27 dimensions of positional and velocity data, with

an option to expand to 29 dimensions by including the x- and y-coordinates of the torso. The reward structure comprises a positive reward for maintaining the robot's health, a forward movement reward, and penalties for excessive control effort and contact forces, encouraging stable and efficient locomotion.

3.2.1.2 HalfCheetah-v4

The HalfCheetah environment features a 2D robot with 9 body parts and 8 joints, designed to simulate a running cheetah. The agent's objective is to apply torques to six key joints (thighs, shins, and feet) to maximize forward velocity. The action space allows for continuous control of these torques within a range of -1 to 1, while the observation space consists of 17 variables capturing positional and velocity data of the cheetah's body parts. The reward function combines a positive component for forward movement and a penalty for excessive torque application, promoting efficient and controlled running behavior. Notably, this environment has no terminal state; the agent's goal is to continue moving forward for as long as possible, maximizing cumulative rewards over time [35].

3.2.1.3 Hopper-v4

The Hopper environment features a one-legged, two-dimensional robot consisting of four main body parts: the torso, thigh, leg, and foot. The goal is to propel the hopper forward by applying torques to the three hinges connecting these parts, resulting in a continuous hopping motion. The action space is defined by a three-dimensional continuous range of torques. The observation space includes 11 variables capturing positional and velocity data. The reward structure combines a healthy reward for maintaining balance, a forward reward for progressing in the positive x-direction, and a control cost penalizing excessive torque use. The total reward is the sum of these components, with the episode ending if the hopper's body falls below a set height, emphasizing the need for stable and efficient movement.

3.2.1.4 Humanoid-v4

The Humanoid environment involves a 3D bipedal robot that simulates human walking by controlling a torso with attached limbs: two legs with three segments each and two arms with two segments. The goal is to move forward rapidly without falling. The environment's action space is a 17-dimensional space representing the torques applied at various joints, while the default observation space, a 376-dimensional space, captures positions and velocities of all body parts. The reward function includes a healthy reward for maintaining an upright stance, a forward reward based on movement in the positive x-direction, and penalties for excessive control forces and external contacts. The total reward combines these components, promoting efficient forward movement while penalizing unsafe or overly forceful actions to encourage stable locomotion [35].

3.2.1.5 LunarLanderContinuous-v2

The LunarLanderContinuous environment is a classic control problem focused on optimizing a rocket's trajectory, where the goal is to land a spacecraft on a designated landing pad at the coordinate (0,0). The action space can be either discrete, with four possible actions (do nothing, fire the left orientation engine, fire the main engine, or fire the right orientation engine), or continuous, allowing for a range of thrust values. The state is represented by an 8-dimensional vector that includes the lander's coordinates, linear velocities, angle, angular velocity, and two boolean values indicating whether each leg is in contact with the ground. Rewards are awarded based on proximity to the landing pad, speed of descent, orientation of the lander, and engine usage, with additional rewards for safe landings (+100 points) or penalties for crashes (-100 points). The agent is encouraged to minimize engine use while ensuring a controlled descent, and an episode is considered successful if it achieves a score of at least 200 points.

3.2.1.6 Walker2d-v4

The Walker2d environment builds upon the Hopper environment by introducing a two-legged robot designed to walk forward instead of hopping. The robot consists of seven main body parts: a torso at the top, two thighs below the torso, two legs beneath the thighs, and two feet supporting the entire structure. The objective is to move forward by applying torques to six hinge joints connecting these body parts. The action space allows for continuous torque control within a range of -1 to 1 across the six joints, while the observation space comprises 17 dimensions, capturing both positional and velocity data for the robot's body parts. This can be expanded to 18 dimensions by including the x-coordinate of the torso. The reward system is composed of three components: a fixed healthy reward for maintaining balance, a forward reward for moving in the positive x-direction, and a control cost that penalizes excessive torque application. The total reward is calculated as the sum of the healthy and forward rewards minus the control cost, encouraging stable and efficient walking behavior.

3.2.2 Implementation Details

The Corrected Uniform Experience Replay (CUER) algorithm was evaluated against the state-of-the-art baseline algorithms, Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC), as well as the Prioritized Experience Replay (PER) strategy. The models were trained and tested across various environments from the Gymnasium suite, including Ant, HalfCheetah, Hopper, Humanoid, LunarLanderContinuous, and Walker2d, using six different random seeds to ensure robust evaluation and minimize the effects of stochastic variability. The original network architectures and hyperparameters specified in the TD3 and SAC papers [15, 16] were followed throughout the experiments.

For both TD3 and SAC, the original architecture was used, where the Actor and Critic networks consist of two hidden layers with 256 neurons each. The Adam optimizer was employed with a learning rate of 3×10^{-4}, and the batch size

was set to 256. Additionally, for TD3, parameters such as the target smoothing coefficient, policy noise, noise clipping, and delay in policy updates were set according to the original implementation [15]. For SAC, parameters like the entropy coefficient (α), target smoothing, and automatic entropy tuning were used as described in the original work [16]. The replay buffer was filled with 25,000 exploration steps generated by random actions to mitigate dependencies on the initialization of environments and network parameters.

Agents were evaluated every 5,000 timesteps, with five test episodes performed using the most recent policy to ensure the performance metrics reflect the latest learned behavior. This periodic evaluation allowed for continuous monitoring of the learning progress and stability of the agents throughout training.

All experiments were conducted on a high-performance computing setup equipped with an AMD Ryzen Threadripper PRO 5975WX CPU with 32 cores and two NVIDIA RTX 6000 Ada GPUs, each with 48 GB of VRAM, to handle the computational demands of training and evaluating multiple deep reinforcement learning models efficiently.

3.2.3 Results for TD3 Algorithm

In this section, we compare CUER with vanilla experience replay and Prioritized Experience Replay (PER). Figures 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 demonstrates how CUER, vanilla experience replay, and PER perform in terms of sample efficiency, convergence speed, and stability of learning. In particular, CUER shows a consistent improvement in learning performance across all environments except Hopper-v4. CUER achieves higher episodic returns faster and with reduced variance compared to both vanilla experience replay and PER.

The returns over episodes in Ant-v4, HalfCheetah-v4, LunarLanderContinuous-v2 and Walker2d-v4 demonstrates the superior performance of CUER by significantly outperforming both vanilla experience replay and Prioritized Experience Replay (PER). In these environments, CUER achieves higher episodic returns

at a much faster rate, suggesting that its experience sampling strategy allows the agent to learn more effectively and efficiently. Notably, CUER consistently converges to optimal policies more quickly and with reduced variance in the returns, indicating a more stable learning process. For example, in the Ant-v4 and Walker2d-v4 environments, CUER shows rapid improvements and maintains superior performance throughout the training, suggesting that the method enhances both convergence speed and final policy quality. The improvements in these environments reflect CUER's ability to balance exploration and exploitation more effectively by dynamically adjusting the sampling probabilities, which leads to more efficient utilization of experience data and robust policy updates.

In certain environments like Humanoid-v4 and Hopper-v4, the performance gains of CUER over the other methods are less pronounced. Although CUER still achieves slightly better returns and maintains lower variance than vanilla experience replay and PER, the differences are minimal, indicating that CUER's advantages may be less impactful in simpler or harder environments due to the bottlenecks resulting from the baseline algorithm itself.

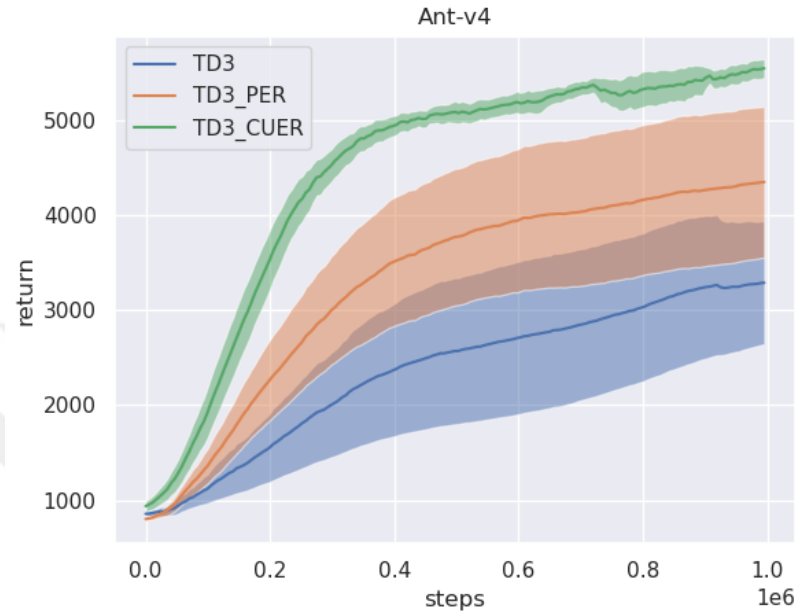

Figure 3.2: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Ant-v4 environment

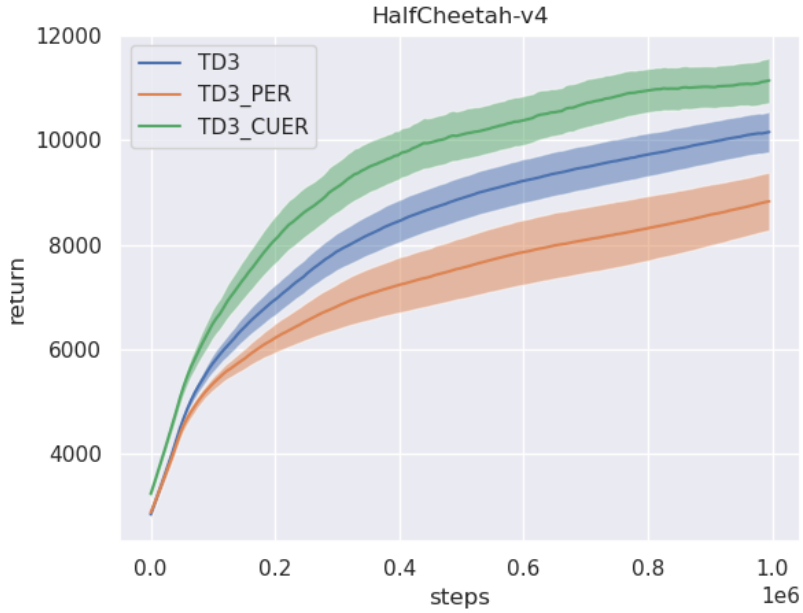

Figure 3.3: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in HalfCheetah-v4 environment

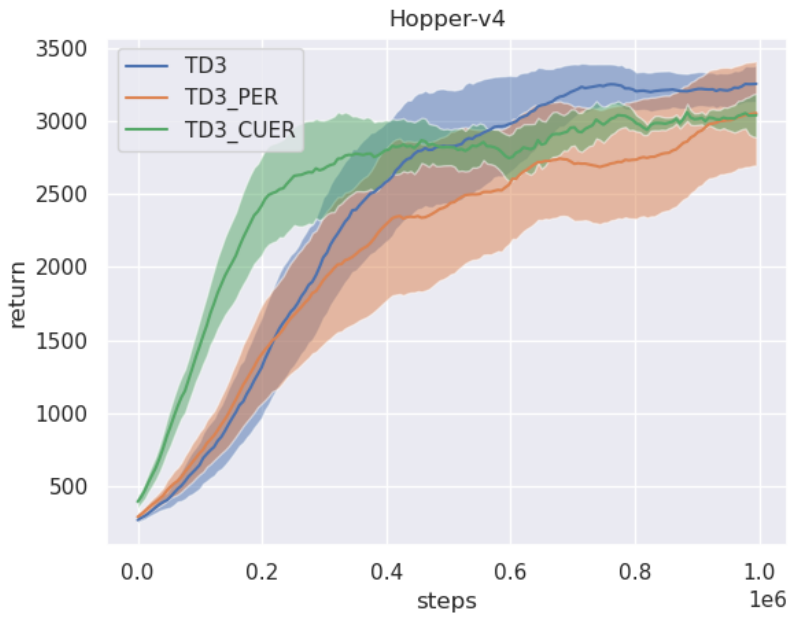

Figure 3.4: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Hopper-v4 environment

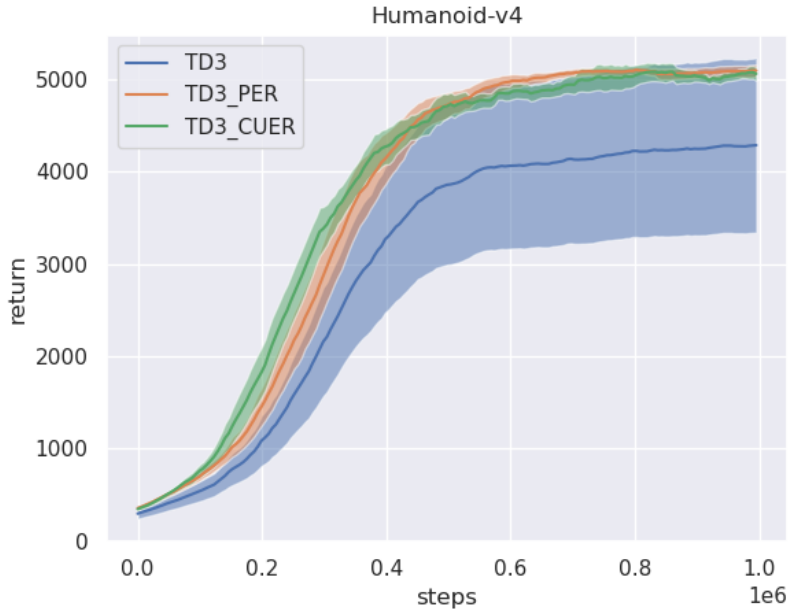

Figure 3.5: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Humanoid-v4 environment

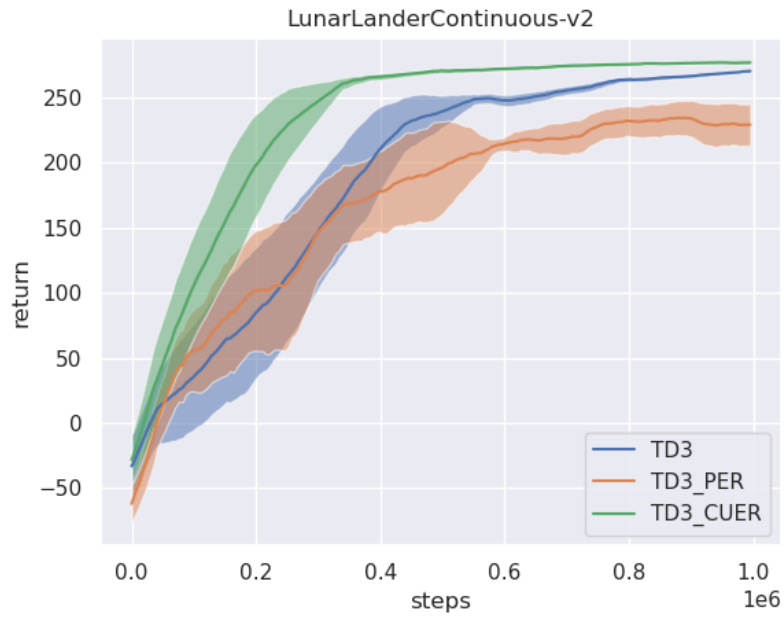

Figure 3.6: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in LunarLanderContinuous-v2 environment

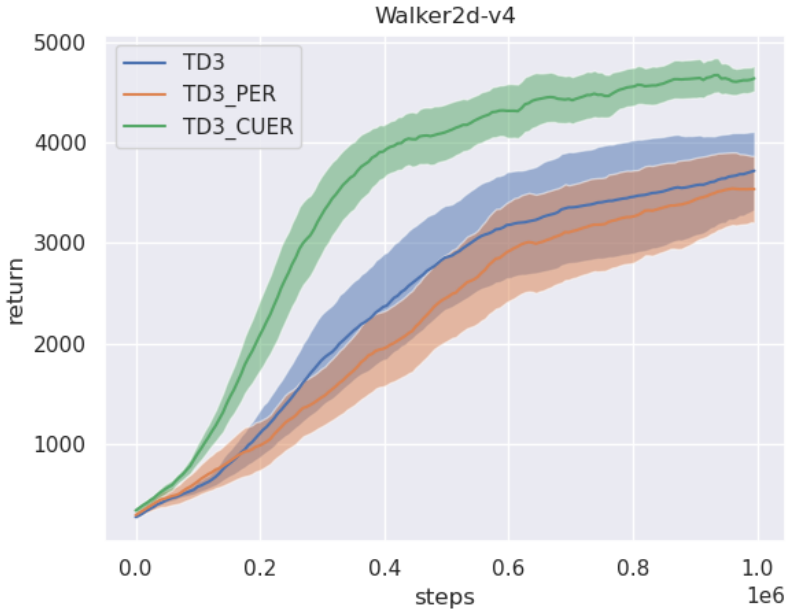

Figure 3.7: Learning progress plot of TD3 for vanilla experience replay, PER and CUER in Walker2d-v4 environment

3.2.4 Results for SAC Algorithm

In this section, we compare the combination of CUER and SAC with the vanilla experience replay and Prioritized Experience Replay (PER). Figures 3.8, 3.9, 3.10, 3.11, 3.12, 3.13 demonstrate CUER consistently outperforms the baseline method and PER in all environments, particularly in terms of sample efficiency, convergence speed, and stability of learning.

In environments such as Ant-v4, HalfCheetah-v4, LunarLanderContinuous-v2 and Walker2d-v4, CUER shows a notable improvement over both vanilla experience replay and PER for SAC algorithm. CUER also achieves higher returns with reduced variance in all environments, reducing the initialization dependency particularly originating from neural networks. Specifically, CUER reaches optimal policies faster and maintains higher returns with lower variance in Ant-v4, HalfCheetah-v4, LunarLanderContinuous-v2 and Walker2d-v4.

In environments like Hopper-v4 and Humanoid-v4, the performance benefits of CUER are less pronounced. While CUER still achieves better returns than vanilla experience replay and PER, the differences are minimal, and the variance remains similar across all methods. This suggests that CUER's advantages may be less impactful in environments where either the learning dynamics are relatively simple, or the challenges are more complex, resulting in a bottleneck in performance gains. In such cases, the improvements brought by CUER may be constrained by the baseline algorithm's inherent limitations, such as the specific learning characteristics or state-action space complexities of the environment.

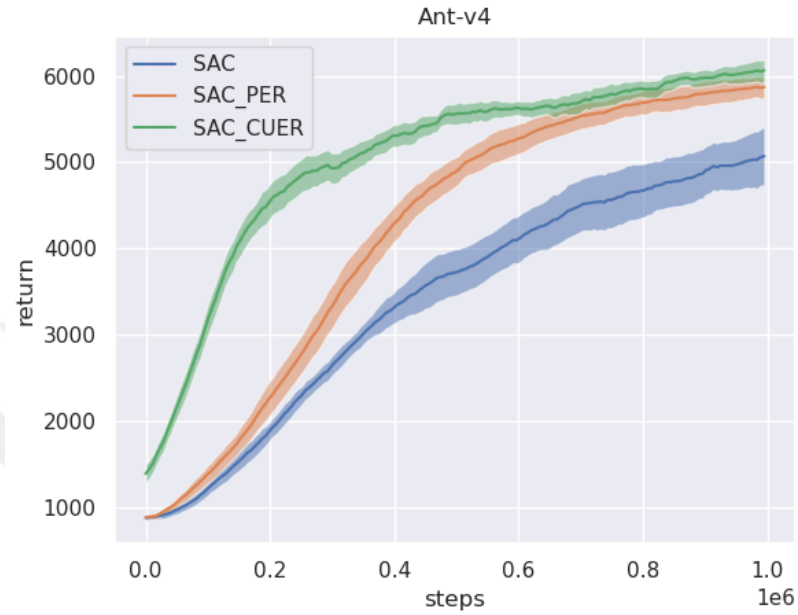

Figure 3.8: Learning progress plot of SAC for vanilla experience replay, PER and CUER in Ant-v4 environment

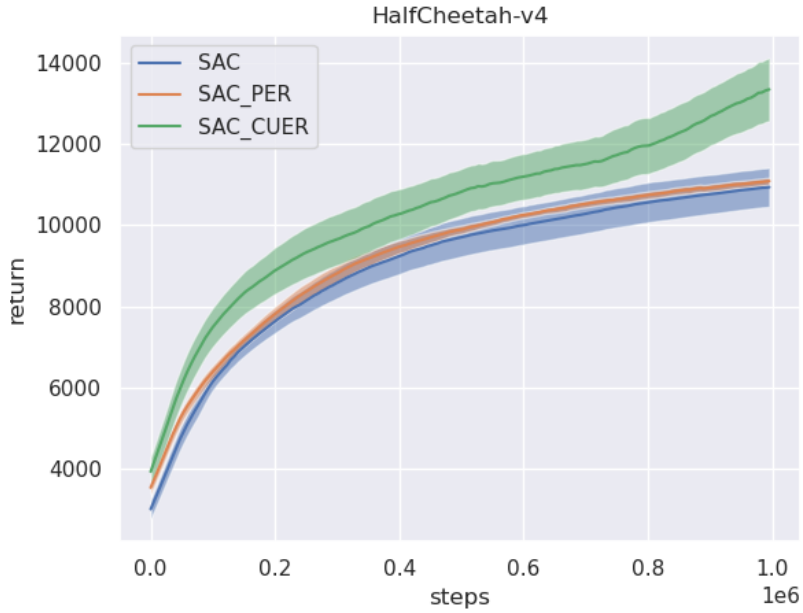

Figure 3.9: Learning progress plot of SAC for vanilla experience replay, PER and CUER in HalfCheetah-v4 environment

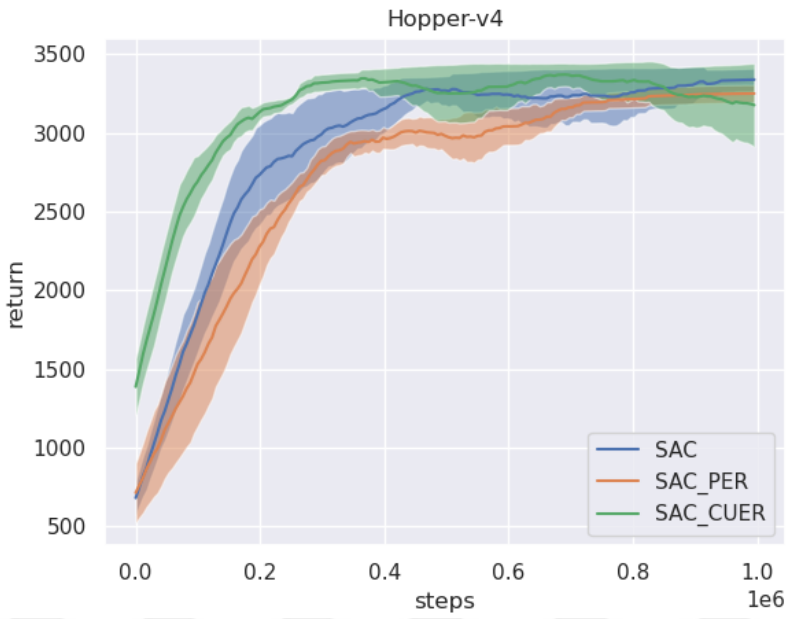

Figure 3.10: Learning progress plot of SAC for vanilla experience replay, PER and CUER in Hopper-v4 environment

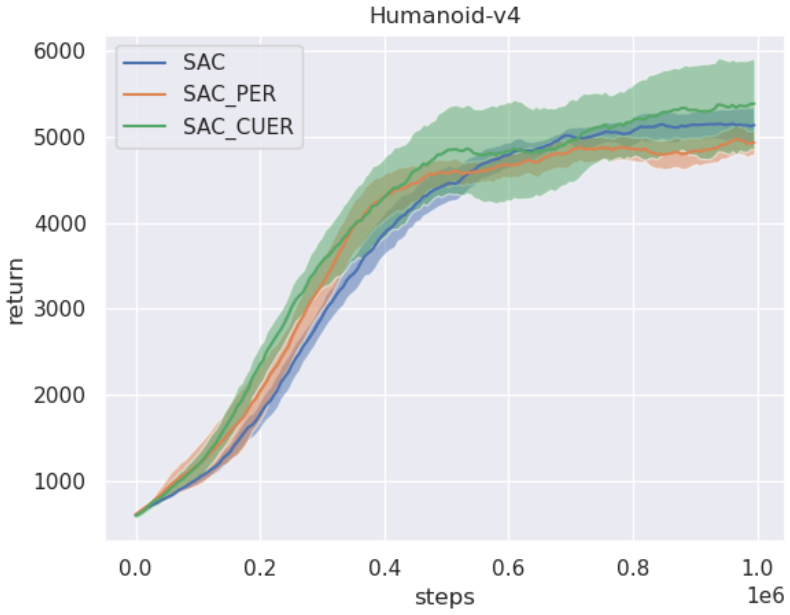

Figure 3.11: Learning progress plot of SAC for vanilla experience replay, PER and CUER in Humanoid-v4 environment

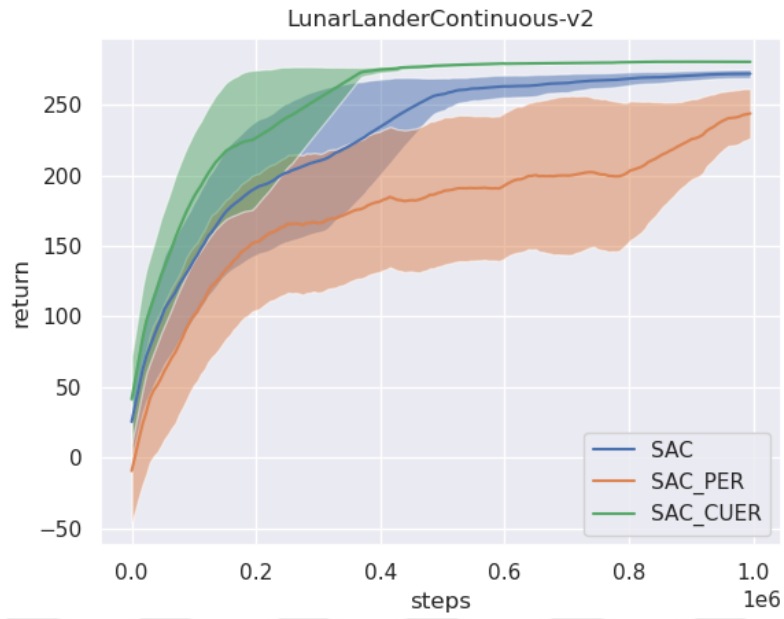

Figure 3.12: Learning progress plot of SAC for vanilla experience replay, PER and CUER in LunarLanderContinuous-v2 environment

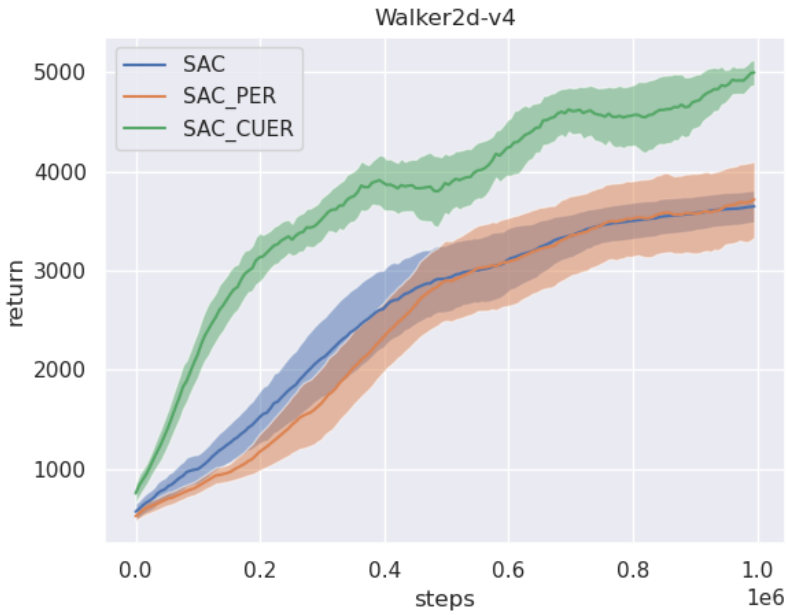

Figure 3.13: Learning progress plot of SAC for vanilla experience replay, PER and CUER in Walker2d-v4 environment

3.2.5 Ablation Studies

To further investigate the impact of CUER on the sampling counts of transition tuples within the replay buffer, we conducted a series of ablation studies. CUER is designed with the core motivation of ensuring that each transition is sampled an equal number of times during training, thereby addressing the inherent bias towards older transitions that tend to be sampled more frequently in traditional experience replay settings. Figure 3.14 demonstrates the equalization provided by CUER during the training by visualizing the sampling counts of each transition during a training with 250000 buffer size. This equalization is crucial to mitigate the over-representation of outdated transitions, which can skew learning and hinder the convergence process. A natural question that arises is whether simply reducing the size of the replay buffer could achieve a similar effect to CUER, by limiting the number of stored transitions and consequently the frequency of sampling older ones. However, our findings demonstrate that this approach does not provide the same benefits. While reducing the buffer size might initially appear to balance the sampling frequency, it overlooks the importance of retaining older transitions to prevent catastrophic forgetting — a common problem in off-policy learning where valuable past experiences are discarded too soon, potentially leading to a significant degradation in policy performance.

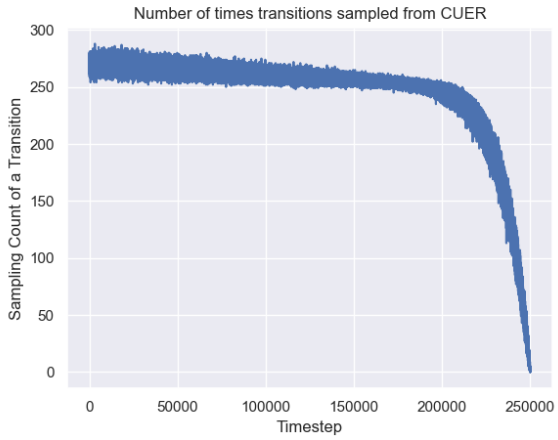

Figure 3.14: Sampling counts of transitions during the training with CUER.

Older transitions remain crucial in the latter stages of training to ensure that the learning agent maintains a well-rounded understanding of the environment, which includes lessons learned from diverse states and actions encountered throughout the training process.

To substantiate this point, we compared CUER against vanilla experience replay with reduced buffer sizes of 100,000 and 250,000. Figure 3.15, 3.16, 3.17, 3.18, 3.19 illustrate that, unlike CUER, simply decreasing the buffer size fails to maintain the necessary diversity of experiences required for robust off-policy learning. This comparison highlights that CUER effectively remedies the bias of high sampling counts of older transitions while still leveraging the importance of those transitions to stabilize learning and improve overall performance. For instance, in the Ant-v4 and Walker2d-v4 environments, reducing the buffer size to 100,000 and 250,000 results in a noticeable drop in performance. The agent demonstrates slower convergence and higher variance in episodic returns compared to CUER. This finding suggests that while reducing the buffer size limits the frequency of sampling older transitions, it also constrains the diversity of experiences necessary for stable learning, leading to degraded performance.

In conclusion, the analysis across different environments confirms that CUER's ability to sample each transition uniformly during training without sacrificing the diversity of experience leads to enhanced performance and robustness. Reducing the replay buffer size alone does not provide the same balance between sampling recency and diversity, underscoring the advantages of CUER in off-policy deep reinforcement learning.

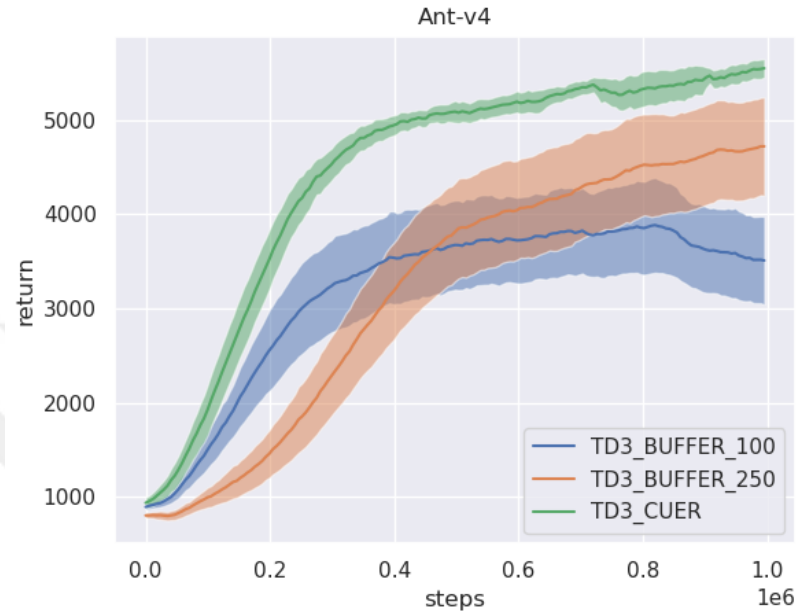

Figure 3.15: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Ant-v4 environment

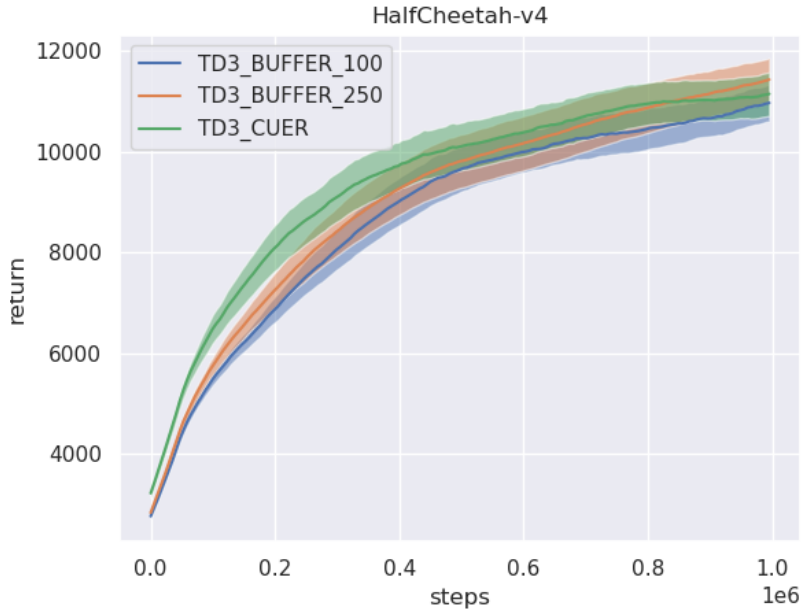

Figure 3.16: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in HalfCheetah-v4 environment

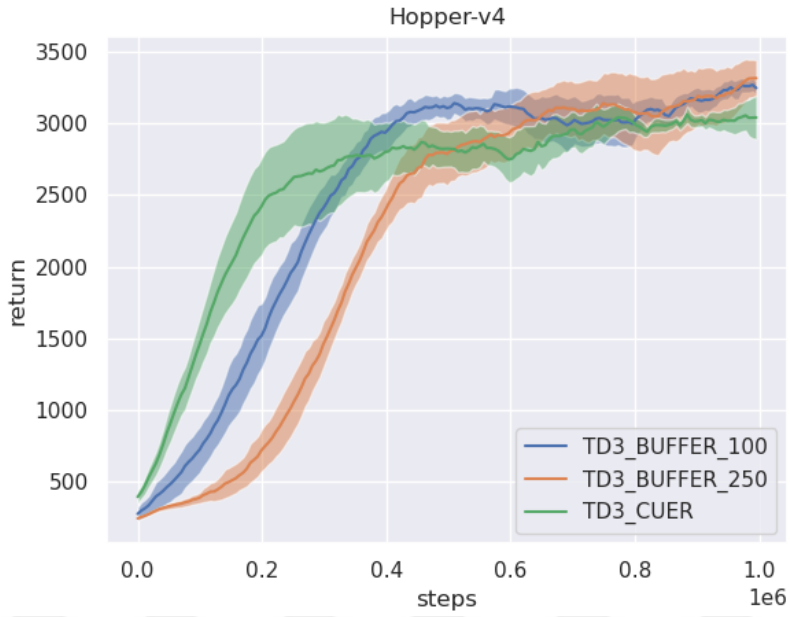

Figure 3.17: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Hopper-v4 environment

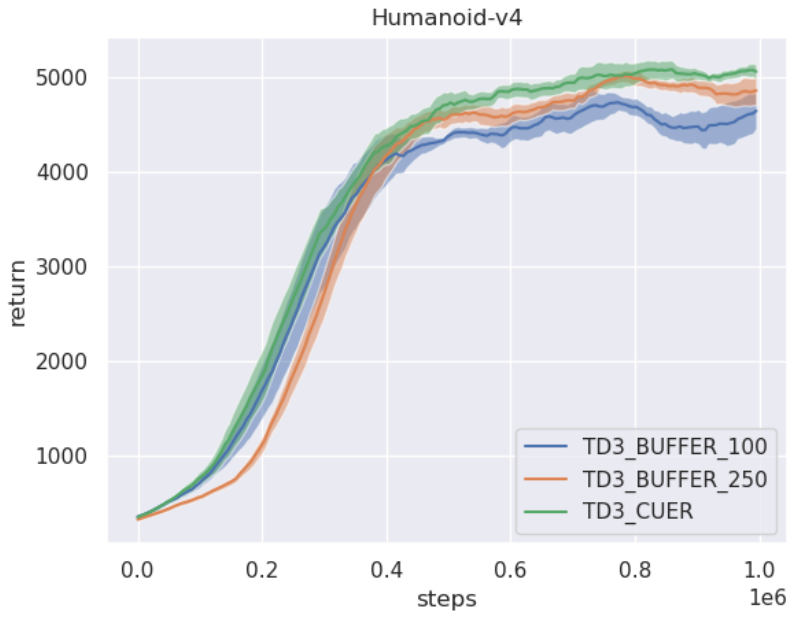

Figure 3.18: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Humanoid-v4 environment

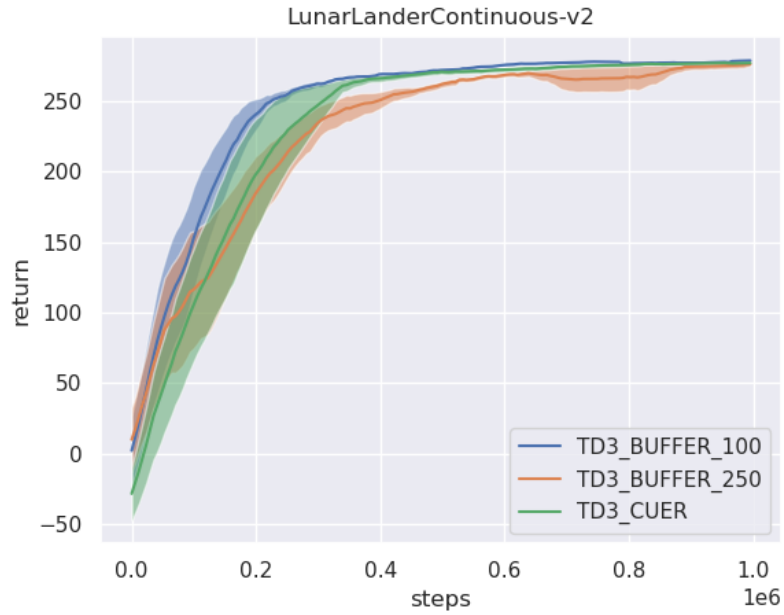

Figure 3.19: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in LunarLanderContinuous-v2 environment

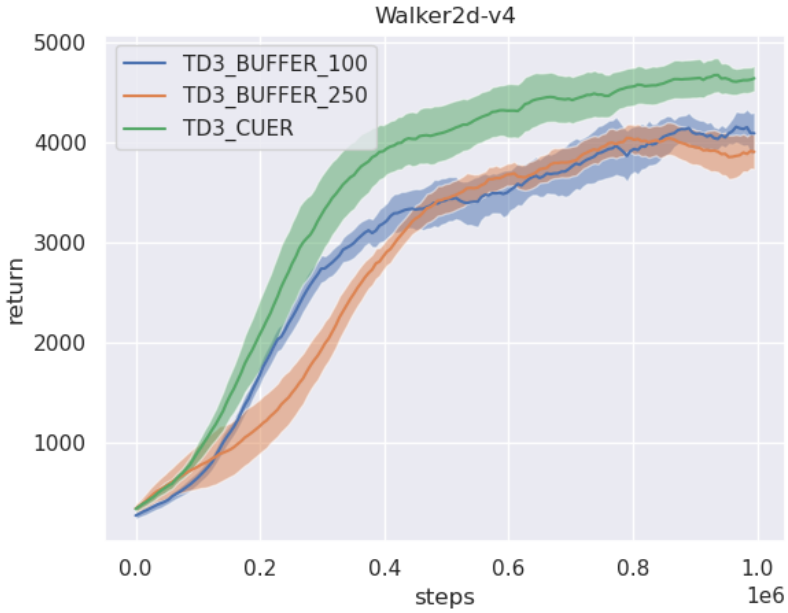

Figure 3.20: Learning progress plot of TD3 agent with vanilla buffer sizes of 100000, 250000 and CUER in Walker2d-v4 environment

Chapter 4

Conclusion

In this thesis, we have addressed the limitations associated with uniform experience replay in reinforcement learning and explored why existing sampling strategies, such as Prioritized Experience Replay (PER), are not well-suited for actor-critic settings. To overcome these challenges, we introduced the Corrected Uniform Experience Replay (CUER), a novel approach grounded in a theoretical framework and enhanced by an innovative batch sampling strategy designed to improve learning efficiency and stability.

We combined CUER with two leading off-policy deep reinforcement learning algorithms, Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC), and conducted extensive experiments to evaluate its effectiveness. The empirical results demonstrate that CUER consistently outperforms both the existing sampling methods, like PER, and the standard uniform experience replay strategies across a wide range of simulation environments. The superior performance of CUER, observed in almost all tested scenarios, highlights its ability to achieve higher episodic returns, faster convergence rates, and more stable learning processes.

Additionally, we conducted ablation studies to examine the impact of buffer size on reinforcement learning performance. The findings revealed that merely

reducing the replay buffer size does not provide the same benefits as CUER. In contrast, CUER maintains an optimal balance by ensuring the relevance of older transitions while effectively mitigating the bias of over-sampling these transitions, thereby preventing catastrophic forgetting and improving overall training efficiency.

This research underscores the critical importance of experience replay mechanisms, the order of transition tuples, and the priority assignments of transitions within the buffer in determining training performance. As a direction for future work, we propose that experience replay sampling strategies should dynamically adapt to the specific needs and training situations of an agent. The priorities and sampling strategies should be continually updated, possibly at each step, to achieve a more effective policy with fewer timesteps. Developing adaptive and context-aware sampling strategies will be crucial in advancing reinforcement learning toward more robust and efficient algorithms.

By highlighting these considerations, this thesis paves the way for new approaches that evolve in response to the specific dynamics of training environments, ultimately contributing to the development of more intelligent and adaptable learning agents.

Bibliography

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

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[3] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[4] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, 11 2019.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[6] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," 2018.

[7] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to sample with local and global contexts in experience replay buffer," 2021.

[8] C. Huré, H. Pham, A. Bachouch, and N. Langrené, "Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis," *SIAM Journal on Numerical Analysis*, vol. 59, p. 525–557, Jan. 2021.

[9] B. Saglam, F. B. Mutlu, D. C. Cicek, and S. S. Kozat, "Actor prioritized experience replay," 2022.

[10] L. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[11] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *CoRR*, vol. abs/1707.01495, 2017.

[12] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2017.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[14] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *CoRR*, vol. abs/1712.01275, 2017.

[15] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[17] D. A. Sofge, "The role of exploration in learning control," 1992.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[19] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[20] R. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 08 1988.

[21] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[23] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.

[24] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, p. 1433–1438, AAAI Press, 2008.

[25] M. Toussaint, "Robot trajectory optimization using approximate inference," vol. 26, p. 132, 06 2009.

[26] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, 2017.

[27] B. D. Ziebart, "Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy," 7 2018.

[28] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," 2018.

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

[30] D. Zha, K. Lai, K. Zhou, and X. Hu, "Experience replay optimization," *CoRR*, vol. abs/1906.08387, 2019.

[31] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to sample with local and global contexts in experience replay buffer," *CoRR*, vol. abs/2007.07358, 2020.

[32] A. S. Yenicesu, F. B. Mutlu, S. S. Kozat, and O. S. Oguz, "Cuer: Corrected uniform experience replay for off-policy continuous deep reinforcement learning algorithms," 2024.

[33] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[34] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[35] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," 2016.