NOVEL EXPERIENCE REPLAY MECHANISMS TO IMPROVE THE PERFORMANCE OF THE DEEP DETERMINISTIC POLICY GRADIENTS 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

Doğan Can Çiçek

September 2022

NOVEL EXPERIENCE REPLAY MECHANISMS TO IMPROVE THE PERFORMANCE OF THE DEEP DETERMINISTIC POLICY GRADIENTS ALGORITHMS

By Doğan Can Çiçek

September 2022

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)

Aykut Koç

Çağatay Candan

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

NOVEL EXPERIENCE REPLAY MECHANISMS TO IMPROVE THE PERFORMANCE OF THE DEEP DETERMINISTIC POLICY GRADIENTS ALGORITHMS

Doğan Can Çiçek
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
September 2022

The experience replay mechanism allows agents to use the experiences multiple times. In prior works, the sampling probability of the transitions was adjusted according to their importance. Reassigning sampling probabilities for every transition in the replay buffer after each iteration is highly inefficient. Therefore, experience replay prioritization algorithms recalculate the significance of a transition when the corresponding transition is sampled to gain computational efficiency. However, the importance level of the transitions changes dynamically as the policy and the value function of the agent are updated. In addition, experience replay stores the transitions generated by the previous policies of the agent that may significantly deviate from the most recent policy of the agent. Higher deviation from the most recent policy of the agent leads to more off-policy updates, which is detrimental for the agent. In this thesis, we develop a novel algorithm, Batch Prioritizing Experience Replay via KL Divergence (KLPER), which prioritizes a batch of transitions rather than directly prioritizing each transition. Moreover, to reduce the off-policiness of the updates, our algorithm selects one batch among a certain number of batches and forces the agent to learn through the batch that is most likely generated by the most recent policy of the agent. Also, previous experience replay algorithms in the literature provide the same batches of transitions to the Actor and the Critic Networks of the Deep Deterministic Policy Gradients algorithms. However, the learning principles of these two cascaded components of a deep deterministic policy gradient algorithm contain dissimilarities in terms of their parameter updating strategies. Due to this fact, we attempt to decouple the training of the Actor and the Critic of the deep deterministic policy gradient algorithms in terms of the batches of transitions that they use during the training of the networks. We develop a novel algorithm, Decoupled Prioritized

Experience Replay, DPER, that enables the agent to use independently sampled batches of transition for the Actor and the Critic of the Deep Deterministic Policy Gradient Algorithms. DPER utilizes Prioritized Experience Replay, PER, and Batch Prioritizing Experience Replay via KL Divergence, KLPER, to decouple the learning processes of the Critic and the Actor, respectively. We combine our algorithms, KLPER and DPER, with the current state-of-the-art Deep Deterministic Policy Gradient algorithm, DDPG, and TD3, and evaluate it on continuous control tasks. KLPER provides promising improvements for deep deterministic continuous control algorithms in terms of sample efficiency, final performance, and stability of the policy during the training. Moreover, DPER outperforms PER, KLPER, and Vanilla Experience Replay on most of the continuous control tasks. DPER outperforms conventional experience replay strategies without adding a significant amount of computational complexity.

Keywords: deep reinforcement learning, experience replay, off-policy learning, prioritized sampling, continuous control.

ÖZET

YENİ TECRÜBE TEKRARI MEKANİZMALARIYLA DERİN DETERMİNİSTİK POLİTİKA GRADYANI ALGORİTMALARININ PERFORMANSINI ARTIRMA

Doğan Can Çiçek
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Eylül 2022

Tecrübe tekrar mekanizması, ajanların tecrübelerini birden çok kez kullanmasını sağlar. Daha önceki çalışmalarda, her bir tecrübenin örnekleme olasılığı, önemlerine göre ayarlanmıştır. Her yinelemeden sonra yeniden oynatma arabelleğindeki her geçiş için örnekleme olasılıklarını yeniden atamak oldukça verimsizdir. Bu nedenle, tecrübe tekrarı önceliklendirme algoritmaları, hesaplama verimliliği elde etmek için karşılık gelen tecrübe örneklendiğinde o tecrübenin önemini yeniden hesaplar. Ancak, ajanın politika ve değer fonksiyonu güncellendikçe geçişlerin önem düzeyi dinamik olarak değişmektedir. Ek olarak, tecrübe tekrarı, ajanın en son politikasından önemli ölçüde sapabilecek olan ajanın önceki politikaları tarafından oluşturulan tecrübeleri depolar. Ajanın en son politikasından daha yüksek sapma, ajan için zararlı olan daha fazla politika dışı güncellemelere yol açar. Bu tezde, her tecrübeye doğrudan öncelik vermek yerine toplu halde örneklenen tecrübelere öncelik veren, KL Iraksaması aracılığıyla Öncelikli Tecrübe Oynatma(KLPER) adında yeni bir algoritma geliştiriyoruz. Ayrıca, literatürdeki önceki deneyim tekrarlama algoritmaları, Derin Deterministik Politika Gradyanları algoritmalarının Aktör ve Kritik Ağlarına aynı tecrübe gruplarını sağlar. Ancak, bir derin deterministik politika gradyan algoritmasının bu iki kademeli bileşeninin öğrenme ilkeleri, parametre güncelleme stratejileri açısından farklılıklar içerir. Bu nedenle, derin deterministik politika gradyan algoritmalarının Aktör ve Kritiğinin eğitimini, ağların eğitimi sırasında kullandıkları tecrübe yığınları açısından ayırmaya çalışıyoruz. Aracının, Derin Deterministik Politika Gradyan Algoritmalarının Aktör ve Kritik için bağımsız olarak örneklenmiş tecrübe yığınlarını kullanmasını sağlayan, Ayrılmış Öncelikli Tecrübe Tekrarı (DPER) adlı yeni bir algoritma geliştiriyoruz. DPER, sırasıyla Kritik ve Aktörün öğrenme süreçlerini ayırmak için Öncelikli Tecrübe

Tekrarı, PER ve KLPER kullanır. Algoritmalarımız olan KLPER ve DPER, mevcut son derin deterministik politika gradyan algoritmaları, DDPG ve TD3 ile birleştiriyor ve sürekli kontrol görevlerinde değerlendiriyoruz. KLPER, eğitim sırasında örnek verimliliği, nihai performans ve politikanın kararlılığı açısından derin deterministik sürekli kontrol algoritmaları için umut verici iyileştirmeler sağlar. Öte yandan, DPER, sürekli kontrol görevlerinin çoğunda PER, KLPER ve Vanilla Experience Replay'den daha iyi performans gösterir. DPER, önemli miktarda hesaplama karmaşıklığı eklemeden geleneksel deneyim tekrar oynatma stratejilerinden daha iyi performans gösterir.

Anahtar sözcükler: derin pekiştirmeli öğrenme, tecrübe tekrarı, politika dışı öğrenme, öncelikli örnekleme, sürekli kontrol.

Acknowledgement

I would like to thank my advisor, Prof. Süleyman Serdar Kozat, for trusting me and giving me a place in his group. During my time in his group, I learned not only how to be a good researcher but also how to be successful throughout my life. I will always be indebted to him for all the precious things he taught me.

I am grateful to my thesis committee members Dr. Aykut Koç and Prof. Çağatay Candan for kindly accepting to review this thesis.

I would like to thank my friends for helping me to shape my character and sharing their valuable friendships with me. I feel very lucky to have so many and valuable friends: Arda Atalık, Çağatay Ateş, Emir Avcı, Serhat Bakırtaş, İsmail Balaban, Recep Bekci, Emir Ceyani, Ali İhsan Çetin, Mete Çoruk, Özkan Demir, Tolga Dimlioğlu, Enes Duran, Serhat Erdoğan, Özgür Eriş, Özen Gümüş, Ahmet Güngör, Yiğit Güvercin, Oğuzhan Karaahmetoğlu, Ozan Karaduman, Erdem Karaosmanoğlu, Kağan Kaya, Önder Kayhan, Anıl Kaplan, Vehbi Kepkep, Asım Kepkep, Berkan Kılıç, Ahmet Kocager, Mehmet Kulkuloğlu, Furkan Mutlu, Berkay Oymak, Mehmet Berkay Ön, Sarp Önal, Yunus Emre Özertaş, Dolunay Sabuncuoğlu, Baturay Sağlam, Çağdaş Saltan, Nurullah Sevim, Onur Soyaslan, Okan Şahin, Gökhan Şahin, Ümitcan Şahin, Onur İlker Şimşek, Alperen Tavşancı, Selim Furkan Tekin, Mümtaz Torunoğlu, Redion Xhepa, Beytullah Yaşar, Mustafa Yılmaz, Erdinç Yukarıkır, and, I would like to thank to the ones that I hope I set a good example for: Emre Abdulkadiroğlu, Baran Aşılıoğlu, Yağız Dilberoğlu, Kaan Durmaz, Efe Karagözlü, Ahmet Berker Koç, İbrahim Orcan Ön. Finally, and most importantly, I am indebted to my family for their support. For being there for me, whenever I need it and for the support they give me. If I had not known that they were behind me in every decision I made and every step I took, I certainly would not have come this far.

I would like to thank Türk Telekom for the support that they provided under the 5G and Beyond Graduate Scholarship Programme.

Contents

List of Figures

List of Publications

This thesis includes content from following publication(s):

1. 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," *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021, pp. 1255-1262

Chapter 1

Introduction

1.1 Preliminaries

Deep Reinforcement Learning techniques have shown notable success on tasks that require sequential decision-making. Deep Reinforcement learning agents reach the superhuman-level performance on ATARI Games [1], continuous control tasks [2], board games [3], and real-time strategy games [4]. Also, deep reinforcement learning approaches solve distinct continuous control tasks such as human-robot collaboration [5], motion planning [6], path planning [7], telerobotics and teleoperation [8], vehicle decision making [9], endoscopic capsule robot navigation [10],and communications [11] with remarkable success.

Classical reinforcement learning algorithms can be divided into two groups with respect to their behavior policy definition: On-Policy reinforcement learning algorithms and Off-Policy reinforcement learning algorithms [12]. We define behavior policy as the policy that the agent follows while exploring the environment and the target policy as the policy that the agent learns to solve the given task. Reinforcement learning agents that learn in an on-policy manner uses the same policy for exploration and exploitation, which means their behavior policy and target policy are the same. On the other hand, off-policy reinforcement

learning algorithms employ different policies for the agent for exploration and exploitation. Therefore, the agent would have two distinct strategies as behavior policy and target policy. In this thesis, we focus on off-policy reinforcement learning algorithms since it enables the agent to reuse its past experiences collected by the agent using its previous policies.

A neural network is a collection of functions that aims to identify underlying links in a data set using a method that imitates how the biological brain functions [13]. Neural networks give a mapping from inputs to outputs and optimize this mapping by a loss function. A neural network with more than or equal to three layers, including the input and output layers, is referred to as a deep learning algorithm [13]. Deep neural networks or deep learning algorithms can be placed as a function approximation tool into classical learning methods. In this sense, coupling reinforcement learning with deep learning enables the agent to learn a parameterized policy and converge to a nearly optimal policy without visiting each state-action pair [12]. On the other hand, feeding the neural network that generates the policy of the agent by temporally correlated inputs violates the i.i.d assumption of the stochastic gradient-based optimization algorithms. Experience Replay tackles the given problem and breaks the temporal correlation by stacking the transitions to a replay buffer, then picking mini-batches among them randomly [14]. Due to this process, the agent learns from the transitions that are collected from various states of the state space of the task. The works show that the utilization of experience replay provides improvements to the agent in terms of sample efficiency and the stability of the policy [1], [15], [16], [17], [18].

Vanilla Experience Replay (Vanilla ER) algorithm samples transitions from the replay buffer randomly. By uniformly sampling the transitions, the algorithm assumes that the importance of each transition is equal to each other. However, the study shows that the strategy on how the agent's experiences are used during the training drastically affects the performance of the agent [19]. Several algorithms are suggested on how experiences should be replayed by assigning sampling probabilities and yield promising results [20], [21], [22], [23], [24], [25], [26], [27], [28].

In this thesis, we introduce a novel experience replay prioritization method, Batch Prioritized Experience Replay via KL Divergence, KLPER. We approach the experience replay prioritization problem by prioritizing the sampled batches of transitions rather than assigning sampling probabilities to the transitions. The main drawback of prioritizing the transitions is that the importance of a transition can significantly change until the transition is sampled again. Therefore, the sampling probabilities of the transitions may not be proportional to their actual importance. In addition, as the policy of the agent changes, the replay buffer contains more off-policy transitions. It has shown that more off-policy updates induce divergence and negatively affect the performance of the agent [24], [29]. Hence, our algorithm forces the agent to learn through the batch of transitions that are more likely generated by the policy of the agent. We assume that each batch has a policy, Batch Generating Policy, that generalizes the past policies of the agent that collected the transitions in the batch. Then, we define the Batch Generating Policy for each batch with respect to the most recent policy of the agent that. We use the KL Divergence between Batch Generating Policy and a multivariate Gaussian distribution with a mean of 0 as a proxy to measure the deviation between the Batch Generating Policy and the most recent policy of the agent.

In this thesis, we also propose a novel experience replay mechanism, Decoupled Prioritized Experience Replay, DPER. In our work, we decouple the training of the Actor and the Critic of the Deep Deterministic Policy Gradient Algorithms in terms of the batches of transitions that they use during the learning process. Our algorithm is usable for every off-policy deep reinforcement learning algorithm. We aim to show that it is possible to improve the performance of a deep reinforcement learning algorithm that has two cascaded neural structures, the Actor and the Critic, by updating them with different batches of transitions. To support our aim, we use two different Experience Replay methods in terms of their prioritization strategy. We highlight that most of the introduced Experience Replay methods cannot be used mutually exclusively since they mainly prioritize transitions rather than batches of transitions. This creates a conflict of interest between combined experience replay algorithms. Evidently, combining two

methods that use the same type of primer strategy while prioritizing transitions may absorb both of the algorithms' advantages. Therefore, DPER mainly follows two experience replay strategies that distinctly prioritize transitions of the replay buffer, Batch Prioritizing Experience Replay via KL Divergence(KLPER) [30], and Prioritized Experience Replay(PER) [25], for the Actor and the Critic, respectively. KLPER prioritizes batches of transitions rather than directly prioritizing each transition sampled to the replay buffer. On the other hand, PER gives a sampling probability that is collected by the agent when it is stored in the replay buffer. We elaborate on the prioritization strategies of these algorithms and our motivation to use these two methods in later sections of this paper. We show that a combination of two different Experience Replay algorithms can yield better results in specific learning tasks when they are used mutually exclusively.

We evaluate KLPER by coupling it with the Deep Deterministic Policy Gradient and the Twin Delayed Deep Deterministic Policy Gradient algorithms. We compare our algorithm with Prioritized Experience Replay and Vanilla ER algorithms, on OpenAi Gym and MuJoCo continuous control tasks [31], [32].

We evaluate our other algorithm, DPER, by coupling it with the TD3 algorithm [33]. We compare our algorithm with Vanilla Experience Replay, PER, and KLPER. We use MuJoCo environments to test our the performance of our proposed algorithm in simulations of real-life related tasks [31], [32].

1.2 Contributions

The main contributions of this thesis are summarized as follows:

1. Prioritize one batch among certain number of batches that sampled from the replay buffer at each iteration.
2. Define Batch Generating Policy with respect to the most recent policy of the agent to obtain the most likely policy that generates the given batch of transitions.

3. Develop KLPER, to enable the agent learn through more on-policy updates. KLPER uses KL Divergence between Batch Generating Policy and the most recent policy of the agent to prioritize batches of transitions.

4. Demonstrate KLPER on 6 different continuous control tasks. Results yield that our algorithm brings significant improvements on particular tasks in terms of final performance and sample efficiency.

5. DPER is the first work that decouples the Actor and the Critic training by using different batch of transitions,

6. Develop DPER to make the Critic-Network learn through transitions that yields more temporal difference error, while the Actor-Network uses more on-policy batches of transition to update its parameters.

7. We show that combination of two experience replay methods can outperform the cases that they used separately.

8. We study and analyze the metrics used for prioritization process of the transitions for both the Actor and the Critic training.

9. We demonstrate that DPER brings significant improvements on continuous control tasks in terms of final performance and sample efficiency.

1.3 Outline

In the remainder of this thesis, the organization is as follows. Chapter 2 gives background information about the algorithms we couple with our method and related works. Chapter 3 covers our proposed algorithms, KLPER and DPER, in detail. Chapter 4 includes the learning environments that we use for evaluating our algorithms and the results that we yield.

Chapter 2

Background

In this chapter, we give a brief introduction to the reinforcement learning framework. We summarize two off-policy deep reinforcement learning algorithms designed for solving continuous control tasks. We also provide details for the experience replay mechanisms.

2.1 Reinforcement Learning

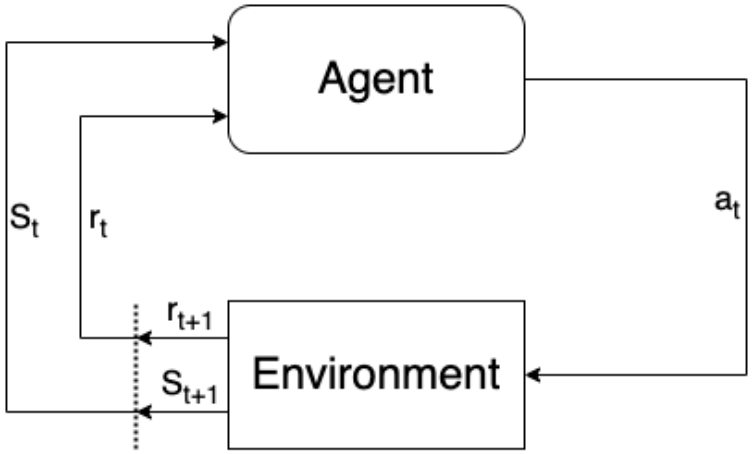

Figure 2.1: Basic Reinforcement Learning Framework

In reinforcement learning, an agent tries to find a cumulative reward maximizing policy by interacting with its environment. Reinforcement Learning tasks are formulated as Markov Decision Processes (MDP) [12]. At each discrete time step t, the agent observes a state $s \in \mathcal{S}$ from the environment. After observing the state information, the agent selects an action from the action space that the reinforcement learning problem have, $a_t \in \mathcal{A}$, with respect to its policy $a_t \sim \pi(a|s_t)$. Then, the agent applies the selected action to the environment and receives the reward, r, and the next state information, $s' \in \mathcal{S}$ from the environment. The environment gives the reward and the next state to the agent according to the environment dynamics $P(s', r|s, a)$. We depict the basic reinforcement learning cycle in Figure 2.1.

The collected information throughout the above cycle defined as a transition, (s, a, r, s'), which is stored to a replay buffer that stores the agent's experiences. The main goal of an agent is to maximize its return, discounted cumulative reward, as defined by:

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

where γ is the discount factor.

2.2 Deep Deterministic Policy Gradient

One of the first deterministic deep reinforcement learning algorithms that tackles tasks with continuous action space is the Deep Deterministic Policy Gradients(DDPG) [2]. The DDPG algorithm has two cascaded deep neural network structures named as the Critic-Network and the Actor-Network. The Critic-Network estimates the Q-value of the given state-action pair as follows:

$$y = C(s, a; \theta), \tag{2.2}$$

where C is the Critic-Network, θ is the parameters of the Critic-Network, and y is the Q-value of the given state-action pairs. The property of the Critic-Network that estimating Q-value by taking state and action information as the input

enables the algorithm to work correctly on continuous action spaces. On the other hand, the Actor-Network controls the agent's behavior with a parametric policy. So, it determines which action should be taken by the agent given the state information:

$$a = A(s; \phi), \tag{2.3}$$

where A is the Actor-Network, ϕ is the parameters of the Actor-Network, and a is the action that is produced by the Actor-Network. In addition to these aforementioned networks, the DDPG algorithm also has one more nested neural network structure named as Target-Network. Target-Network has the Actor-Target-Network and the Critic-Target-Network, which are initialized identically to the Actor-Network and the Critic-Network, respectively, at the beginning of the training. In the DDPG algorithm, the Critic-Network and the Actor-Network are updated sequentially. In the conventional update method for these networks, first, a one-step temporal difference error is calculated, and square of the one-step temporal difference error is defined as the loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - C(s, a; \theta)]^2, \tag{2.4}$$

where $\mathcal{D}$ is the experiences that collected by the agent., y is defined as:

$$y = r + C'(s', A(s'; \phi'); \theta'), \tag{2.5}$$

where C' is the Critic-Target-Network, A is the Actor-Target-Network, ϕ', and θ' are the parameters of the Actor-Target-Network and the Critic-Target-Network, respectively. The output of the Critic-Network can be assumed as the objective function of the agent, and it represents the discounted cumulative reward of the agent at a given state by following the policy that is parameterized by the Actor-Network:

$$J(\theta, \phi) = \mathbb{E}_{s\sim\mathcal{D}}[C(s, a; \theta)|_{a=A(s;\phi)} A(s; \phi)], \tag{2.6}$$

Therefore, the Actor-Network is updated by taking derivative of the objective function respect to the parameters of the Actor-Network:

$$\nabla_\phi J(\theta, \phi) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_a C(s, a; \theta)|_{a=A(s;\phi)} \nabla_\phi A(s; \phi)], \tag{2.7}$$

It has been shown that bootstrapping from the same neural network that estimates the Q-value significantly harms the learning process [34]. To tackle

this problem, the DDPG algorithm utilizes Target-Networks. The Actor-Target-Network and the Critic-Target-Network are softly updated by using the Actor-Network and the Critic-Network:

$$\phi' = \tau\phi + (1-\tau)\phi', \quad \theta' = \tau\theta + (1-\tau)\theta', \tag{2.8}$$

where τ is the rate of the soft update.

2.3 Twin Delayed DDPG

Studies suggest that even though the DDPG algorithm performs well on continuous control tasks, it suffers from an overestimation problem [33], [2]. Overestimation problem is defined as estimating the Q-value of a given state-action pair, (s, a), while following a policy, ϕ, higher than it should be:

$$C_\pi(s, a) > Q_\pi(s, a), \tag{2.9}$$

where C_π is the estimated Q-value of the given state-action pair while following the policy π, and Q_π is the true Q-value of the given state-action pair while following the policy π. Despite the presence of the Target-Networks, in some cases, bootstrapping on overestimated state-action pairs dominantly accelerates the overestimation problem and cripples the learning process. Moreover, it may lead to the agent instantly forgetting its policy in the middle of the training. TD3, remedies this problem by adding one more Critic-Network to the neural network structure [33]. TD3 algorithm formulates the target value for the Q-value of a given state-action pair as follows:

$$y = r + \gamma \min_{i=1,2} Q'_i(s', \psi'(s'; \phi'); \theta'_i), \tag{2.10}$$

and the authors call this procedure as Clipped Double Q-Learning [33]. This advancement enables the TD3 to outperform the DDPG in continuous control tasks. The TD3 solves tasks that the DDPG cannot find any decent policy.

2.4 Experience Replay Mechanism

Neural Networks yield much better results when the training dataset includes mostly uncorrelated samples. However, in reinforcement learning, the agent receives state, and next state information is overlapped for successive transitions, i.e., the agent's next state, s', at time step t, is the same as state, s, at the time step $t+1$. Then, training deep reinforcement learning is challenging since the agent updates its policy and value function using heavily correlated samples from the environment.

The experience replay mechanism remedies the given fact by adding a cyclic replay buffer to the algorithms. The first experience replay method stores the experiences that the agent collects while exploring the environment to the replay buffer as transitions. Then, the agent updates its parameters using the samples selected from the replay buffer to break the correlation between samples that using for the training at a time. The most primitive method samples transitions from the replay uniformly.

Uniformly sampling transitions from the replay buffer indirectly mean that each transition is equally important for the agent's learning process. However, there are studies that change each transition's sampling probabilities to distinguish the transitions that are more and less beneficial transitions for the learning process of the agent.

Prioritized Experience Replay is one of the most well-known techniques used for assigning a different level of importance for the transitions [25]. PER increases the tendency to feed the value function of the deep reinforcement learning algorithm for the transitions collected while the agent encounters unexpected outcomes. PER measures the unexpectedness level of a transition by using temporal difference error as a proxy. temporal difference error can be defined as follows:

$$\delta = r(s, a, s') + \gamma C'\left(s', a'; \theta'\right) - C(s, a; \theta), \tag{2.11}$$

where θ and θ' represent the value and target value network parameters, respectively.

Transitions in the replay buffer can be sampled by following heuristic rules. Focused Experience Replay introduces an alternative probability distribution, half normal distribution, to increase the sampling probabilities of the recently added transitions to the replay buffer [35]. Hindsight Experience Replay is an experience replay method that provides sub goals for the agent to divide the main task into smaller tasks through experience replay. It designed for the tasks that has sparse reward signals [36].

Experience Replay is also can be formulated as a learning problem that proceeds parallel with the training of the deep reinforcement learning agent [28]. The priorities of the transitions stored in the replay buffer can change dynamically as the training process continues. Neural Experience Replay Sampler is a method that handles the experience replay prioritization by adding a parameterized replay policy [26]. This method gives scores to the transitions when the corresponding transition is collected. The score given to the transition is proportional to its importance level. The importance level is determined according to the replay policy. The replay policy considers the information that transition carries, such as state, action, the reward of the transition, temporal-difference error, and timestep while scoring each transition.

Chapter 3

Batch Prioritizing Experience Replay via KL Divergence (KLPER) and Decoupled Prioritized Experience Replay(DPER)

In this chapter, we discuss the drawbacks that may encountered by experience replay prioritization by following the procedure that the PER algorithm proposes. We also discuss the motivation behind the decoupling of the Actor-Network and the Critic-Network training by using different batches of transition for updating these networks. Lastly, we elaborate on the advantages of reducing off-policiness of a learning algorithm. Also, we provide the details of our algorithm, DPER. We give more details on batch selecting strategy of our algorithm for the Actor-Network and the Critic-Network.

3.1 Batch Prioritizing Experience Replay via KL Divergence

In this section, we mention the problems that KLPER aims to tackle. We define the Batch Generating Policy, one of the critical components of the algorithm. Then, we give more details about the batch selection process among candidate batches.

3.1.1 Motivation

Increasing sampling probability of the specific transitions over the other ones may lead to undesirable updates on the Actor and the Critic networks. The likelihood of having these unwanted updates is proportional to the heaviness of the prioritization since heavy prioritization causes more Off-Policy Updates [29].

The sampling probability of a transition reflects the importance of the transition. Each transition's contribution to the learning process depends on the policy of the agent and the value network that the agent uses. To quantify the importance of a transition, one measure should be defined. For instance, PER uses Temporal Difference error for that purpose [25]. Furthermore, the importance of the transitions changes during the training since the policy of the agent or the value network that the agent uses is updated after each iteration. Then, to properly arrange the importance of the samples in the buffer, one should span the whole replay buffer and recalculate the sampling probabilities, which is infeasible in terms of computation after some point since the number of samples in the buffer increases rapidly. PER uses a practical solution for the given problem, recalculates a transitions sampling probability when it is sampled [25]. In that case, the expected sampling period of a prioritized transition is calculated as follows:

$$T_s = \frac{1}{P_i b} \tag{3.1}$$

where P_i is the ith transition's sampling probability and b is the mini-batch size.

This procedure assumes that the importance of a transition remains the same until it is sampled again. However, a desirable transition may become indifferent or even adversarial at the latter stages of the training for the agent and vice versa [24]. Due to the aforementioned issues, an algorithm that prioritizes transitions may lead to an undesirable training process. Then, the Vanilla ER algorithm that samples transitions may outperform a method that prioritizes the samples in the replay buffer [29].

In reinforcement learning, the deadly triad is defined as the combination of function approximation, bootstrapping, and off-policy learning. An algorithm that includes these three properties may suffer from unbounded value estimates, which deteriorates the learning process of the agent [29]. Function Approximation is the most indispensable component of reinforcement learning among the properties of the deadly triad because when state and action spaces of the reinforcement learning task are huge, then visiting all state-action pairs becomes infeasible, especially in the continuous domain. One may use Monte Carlo learning instead of Temporal Difference learning, then discard the Bootstrapping. However, Monte Carlo learning requires long trajectories that end with a terminal state. Tasks that have no termination conditions cannot be properly solved by Monte Carlo learning. If the On-Policy learning is chosen over Off-Policy learning, the agent cannot use the experiences generated from its past policies. Then, the heavily correlated transitions negatively affect the neural network training [29]. The final component of the deadly triad, Off-Policy learning, can be softened by changing the sampling probabilities of the transitions and the performance of the modified algorithm performs better on the learning tasks [29].

In this thesis, we introduce a novel experience replay prioritization algorithm, KLPER, which reduces the off-policyness of the updates at each iteration for Deep Deterministic Policy Gradient algorithms. Our approach selects one batch among candidate batches rather than assigning sampling probabilities to the transitions in the replay buffer.

3.1.2 Batch Generating Policy

Behavior policy is defined as a mixture of an exploration noise and the target policy of the agent, which is parameterized by the Actor network, for DDPG and TD3 algorithms. The replay buffer of an agent includes transitions gathered by the past policies of the agent. We attempt to find the most likely policy that generates the sampled batch of transitions with respect to the most recent policy of the agent. We call that policy as Batch Generating Policy and denote it as ω. We remark that the policy used for generating each transition becomes intractable after the transition is stored due to the exploration noise term of the behavior policy. Thus, we assume that the Batch Generating Policy is stochastic. We elaborate on how we build the Batch Generating Policy for the remaining part of this subsection.

Feedforwarding the states in the batch of transitions, $\boldsymbol{S}^{b\times m}$, to the Actor network yields the actions, $\hat{\boldsymbol{A}}^{b\times l}$ that the agent act in these states respect to its most recent policy:

$$\hat{\boldsymbol{A}}^{b\times l} = \psi(\boldsymbol{S}^{b\times m}; \phi), \tag{3.2}$$

where b is the mini-batch size, ψ is the Actor network, l and m are the number of dimensions that action and state space have, respectively. The difference between actions in the batch and actions that would be taken by the agent's most recent policy, $\dot{\boldsymbol{A}}^{b\times l}$, represents the deviation between the current policy of the agent and previous policies of the agent that were used to generate experiences.

$$\dot{\boldsymbol{A}}^{b\times l} := \hat{\boldsymbol{A}}^{b\times l} - \boldsymbol{A}^{b\times l}, \tag{3.3}$$

where $\boldsymbol{A}^{b\times l}$ is the actions stored in the transitions of the sampled batch. The exploration noise choice affects the behavior policy of the agent. The works show that using Gaussian noise as the exploration noise rather than Ornstein-Uhlenbeck noise [37] does not decrease the performance of the DDPG algorithm [33], [38]. Hence, We choose Gaussian noise,as the exploration noise for both algorithms.

We remark that Batch Generating Policy is stochastic, and we defined it as a probability distribution. The mean of the distribution can be formulated as the

difference between the actions of the sampled transitions and the actions that the agent would produce with respect to states of the corresponding transition:

$$\mu_{\omega}^{1\times l} = \frac{1}{b}\sum_{i\in b} \dot{\boldsymbol{A}}_{ij}^{b\times l}, \tag{3.4}$$

where i is the ith element of the batch and j is the jth dimension of the action space. We define the covariance matrix of the distribution as follows:

$$\Sigma_{\omega}^{l\times l} = \frac{1}{b-1}\sum_{k\in b}(\dot{\boldsymbol{a}}_{k}^{1\times l} - \mu_{\omega}^{1\times l})^{\top}(\dot{\boldsymbol{a}}_{k}^{1\times l} - \mu_{\omega}^{1\times l}), \tag{3.5}$$

where $\dot{\boldsymbol{a}}_{k}^{1\times l}$ is the kth row of the $\dot{\boldsymbol{A}}^{b\times l}$ matrix. We define the shape of the distribution as the multivariate Gaussian by Maximum Entropy Principle. Finally, we obtain Batch Generating Policy:

$$\omega \sim \mathcal{N}(\mu_{\omega}^{1\times l}, \Sigma_{\omega}^{l\times l}). \tag{3.6}$$

3.1.3 Choosing Batches with KL Divergence

In this section, we give more details on how KLPER scores and chooses one batch among candidate batches.

Firstly, at each training timestep t, KLPER samples N candidate batches before updating the parameters of the Actor and the Critic networks. Then, the algorithm derives Batch Generating Policy for each sampled batch. It uses KL Diverge to measure the similarity between the policy of the agent and the Batch Generating Policy. The Actor networks for the DDPG and TD3 algorithms yield deterministic policies. However, we remark that the transitions are generated by following the behavior policy of the agent. Therefore, we define the target distribution for the KL Divergence as a multivariate Gaussian distribution with mean 0 and variance $\sigma\mathbb{I}$. We refer KL score for each batch with κ and KLPER calculates the KL Score as follows:

$$\kappa = \mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\mu_{\omega}, \Sigma_{\omega}) \| \mathcal{N}(0, \sigma\mathbb{I})), \tag{3.7}$$

where $\mathbb{I}$ is the identity matrix. KLPER selects the batch that yields the minimum KL score among candidate batches to provide the learning algorithms more on-policy updates at each iteration. We provide KLPER in Algorithm 1.

Algorithm 1 KLPER

Initialize batch size b, replay buffer $\mathcal{D}$
Initialize M for delayed policy updates
Initialize number of candidate batches N
Initialize σ for the target distribution
for $t = 1$ **to** T **do**
  Observe s_t Choose action $a_t \sim \pi_\phi(a|s_t) + \epsilon$
  Observe reward r_t and new state s'_t
  Store transition (s_t, a_t, r_t, s'_t) in $\mathcal{D}$
  Sample N batches from $\mathcal{D}$
  for $n = 1$ **to** N **do**
    Calculate κ_n for the batch (Eq. 14)
  end for
  Select the batch that yields minimum κ
  Update the weights of the Critic network
  if k mod M **then**
    Update the weights of the Actor network
    Update the weights of the Target networks
  end if
end for

3.2 Decoupled Prioritized Experience Replay

In this section, we provide the details of our algorithm, DPER. Also, we give more details on batch selecting strategy of our algorithm for the Actor-Network and the Critic-Network.

3.2.1 Motivation

Prioritized Experience Replay assigns sampling probabilities to the transitions. We remark that each transitions' sampling probability is proportional to the temporal difference error that they produce:

$$\delta = r(s, a, s') + \gamma C'\left(s', a'; \theta'\right) - C(s, a; \theta), \tag{3.8}$$

Intuitively, this procedure increases the tendency to train the Critic-Network through the transitions that the Critic-Network incorrectly estimates the Q-value

of that they yield. Then, it accelerates the learning process of the Critic-Network. However, it is not the case when this idea is applied to the Actor-Network update. To elaborate on this main drawback of Prioritized Experience Replay, Fig. 1 illustrates the main cascaded actor-critic network of a DDPG algorithm which is designed for a simple task that has two dimensions and one dimension in state and action spaces, respectively. Dashed and straight lines represent the parameters of the Critic-Network and the Actor-Network, respectively. As we discuss the parameter update sequence later, the DDPG and TD3 algorithms updates their Critic-Networks, firstly. Then, the Actor-Networks of these algorithms update their parameters to improve the objective function. During the optimization step, the algorithms use the parameters of the Critic-Network that connect the action produced by the Actor-Network to the Q-Value calculation. In our simple DDPG network illustration, in Fig. 1, these parameters correspond to the parameter $a1q1$. Therefore, the Actor-Network can be updated correctly when the Critic-Network optimization is decent for the given transition set. Conversely, Prioritized Experience Replay suggests transitions that the Critic-Network produces high temporal difference error to the Actor-Network. Then, updating the Actor-Network with this learning procedure deteriorates the Actor-Network optimization.

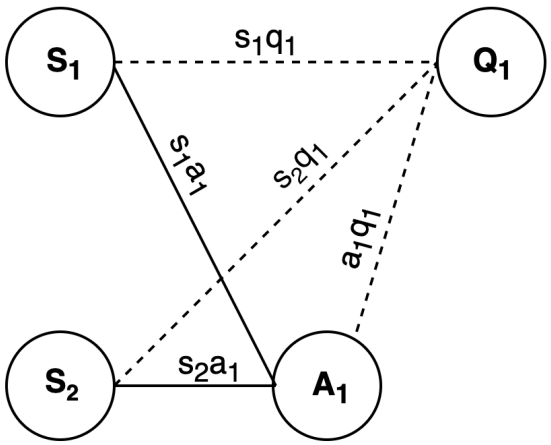

Figure 3.1: Basic DDPG Neural Network structure

3.2.2 Training the Actor-Network and the Critic-Network with Different Batch of Transitions

The DPER trains the Actor-Network by following the KLPER algorithm. To decouple the Actor and the Critic network training, we choose the proportional Prioritized Experience Replay method as the transition sampling strategy of our algorithm's Critic-Network training phase with minor changes [25]. Proportional Prioritizes Experience Replay mechanism adjusts sampling probabilities of each transition that stored proportionally to the magnitude of the temporal difference error that they yield:

$$\psi = |r(s, a, s') + \gamma C'(s', a'; \theta') - C(s, a; \theta)|, \tag{3.9}$$

We denote the magnitude of the temporal difference error as ψ. In our algorithm, we assign a sampling probability to the transition right after it is gathered by the agent. Reassigning sampling probabilities of each transition after each time step is impractical when the replay buffer includes a substantial amount of transitions. Therefore, our algorithm does not change the sampling probabilities of the transitions until they are sampled from the replay buffer as the proportional Prioritized Experience Replay algorithm does. One main drawback of this procedure can occur at the early stages of the training. Assume that one transition that is stored to the buffer at the beginning of the replay outputs a minuscule temporal difference error. It would have a tiny sampling probability that prevents it from being resampled again, and it may occupy the replay buffer until being overridden by another transition. However, the importance level of this transition may change during the training in terms of the magnitude of the temporal difference error that it produces since the Critic-Network and the Critic-Target-Network parameters change during the training. Therefore, to prevent our algorithm for such cases, we use bias correction parameter, α and prioritize transitions as follows for the Critic-Network training by following the process that the proportional Prioritized Experience Replay method proposes:

$$P(i) = \frac{\psi_i^\alpha}{\sum_k \psi_k^\alpha}, \tag{3.10}$$

where, $P(i)$ is the sampling probability of the ith transition in the replay buffer, ψ_i and ψ_k is the magnitude of the temporal difference error of the ith and kth transitions in the replay buffer[25].

Algorithm 2 DPER

Initialize batch size b, replay buffer $\mathcal{D}$
Initialize M for delayed policy updates
Initialize number of candidate batches N
for $t = 1$ **to** T **do**
  Observe s_t Choose action $a_t \sim \pi_\phi(a|s_t) + \epsilon$
  Observe reward r_t and next state s'_t
  Store transition (s_t, a_t, r_t, s'_t) in $\mathcal{D}$
  for $i = 1$ **to** b **do**
    Sample batch of transitions B_i
    Compute TD-error
    Update transition priority
  end for
  Update the weights of the Critic-Network with B
  if k mod M **then**
    Sample N batches from $\mathcal{D}$
    for $n = 1$ **to** N **do**
      Calculate κ_n for the batch
    end for
    Select the batch, B, that yields minimum κ
    Update the weights of the Actor-Network with B
    Softly update the weights of the Actor network
  end if
  Softly update the weights of the Target networks
end for

Chapter 4

Experiments

In this chapter, we elaborate on the learning environments that we compare the performance of the DPER with TD3, PER, and KLPER. We discuss the metrics that we collect during the training, the average magnitude of the temporal difference error and KL scores, and represent learning curves of DPER, KLPER, PER, and Vanilla ER when they are combined with the DDPG and the TD3 algorithms.

4.1 Learning Environments

4.1.1 InvertedPendulum-v2

InvertedPendulum-v2 learning environment consists of a four-dimensional state space. In this setting, a linearly mobile cart with one end fastened to a pole and the other end free is used. By exerting forces on the cart, which may be moved in any direction, the object is to balance the pole on top of it. The action space of this environment has only one dimension, which represents the force applied to the pole. The episode ends if the angle between horizon and pole is lower than a certain threshold.

4.1.2 Reacher-v2

"Reacher" is a robot arm with two joints. The objective is to get the robot's fingertip end effector as close as possible to a randomly placed target. The state space of this task includes 10 dimensions. Two dimensions make up the action space. The torques applied at the hinge joints are represented by an action applied to the environment's actor.

4.1.3 LunarLanderContinuous-v2

LunarLanderContinuous-v2 learning environment consists of a two-dimensional space vehicle that aims to land on a landing pad whose location is constant for every trial. The task is safely land the vehicle by pumping air to the left, right, or downwards directions with respect to the vehicle. Observation space includes information such as the location, angular velocity, and linear velocity of the space vehicle. One episode is terminated if the vehicle crashes or lands on the landing pad safely. Also, the episode finishes if the vehicle spends more than 1,000 time steps in one episode.

4.1.4 BipedalWalker-v3

BipedalWalker-v3 environment has one two-dimensional biped robot. The task is to provide a policy that enables the robot to consistently and quickly move forward until the end of the given path. The observation vector carries information such as hull angle speed, horizontal speed, angular velocity, vertical speed, the position of joints and joints, angular speed, legs contact with the ground, and ten lidar rangefinder measurements. Terminal conditions for this learning environment are falling or reaching the end of the path.

4.1.5 Hopper-v2

The Hopper-v2 environment consists of a one-legged robot that tries to cover a given distance without falling. State-space of this environment has 20 dimensions, such as the information that comes from joint angles, joint velocities, and the coordinates of the center of mass [39]. Being its body lower from a designated level of altitude is the termination condition of this environment.

4.1.6 HalfCheetah-v2

The Half-Cheetah environment consists of a two-dimensional biped robot. The task for this environment is similar to the BipedalWalker-v3 environment, except there is no termination condition for the environment. The agent is responsible for going forward as much as possible [39]. An episode is terminated after 1,000 time steps. Observation collected from the environment has information such as joint angles, joint velocities, and the coordinates of the center of mass.

4.1.7 Walker2d-v2

The Walker2d-v2 is a learning environment that includes a leg-shaped robot. The robot has seven links: two legs, a torso, and six actuated joints. Finding a decent policy for the given task is more challenging than Hopper-v2 environment. Avoiding falling is more demanding than in the Hopper-v2 environment in Walker2d-v2 environment [39]. Observation space includes information such as joint angles, joint velocities, and the coordinates of the center of mass.

4.1.8 Ant-v2

The Ant-v2 environment consists of a four-legged robot that tries to cover a given distance without falling. Unlike the Walker2d-v2, the environment state-space of

this problem is relatively big. It has 125 dimensions in its observation space, such as the information extracted from joint angles, joint velocities, coordinates of the center of mass, a (usually sparse) vector of contact forces, and the rotation matrix for the body [39]. Being its body lower from a designated level of altitude is the termination condition of this environment.

4.2 Experiment Results for KLPER

4.2.1 Implementation Details

We note that our algorithm compared with Vanilla ER and PER. To make a fair comparison, we have run the algorithms by using 5 different random seeds. For each experience replay method, we use the same architecture and hyperparameters for DDPG and TD3 algorithms. Also, we take the network architectures from the original implementations of the DDPG and TD3 algorithms [2], [33].

DDPG has two hidden layer neural networks for both the Actor and the Critic networks, which include 400 and 300 neurons, respectively. For TD3, we use two hidden layer networks that have both 256 neurons, and the network architecture for Actor and Critic are the same. We use Adam as the optimizer of the networks for both DDPG and TD3 [40]. The learning rates of the Actor and the Critic networks are 1×10^{-4} and 3×10^{-4}, respectively, for the DDPG. The learning rate of both Actor and Critic networks for the TD3 algorithm was 1×10^{-3}. We set the mini-batch size to 64 and 256 for DDPG and TD3. We observe that the DDPG algorithm sticks at the local optima when the number of exploration steps is insufficient. Therefore, we fill the replay buffer with 10,000 transitions gathered by the agent while acting randomly for the DDPG algorithm for each experience replay method. The number of exploration steps set to 25,000 for TD3 algorithm.

We choose N, an adjustable parameter for the algorithm, as 4 for DDPG and 8 as TD3. We choose σ as 0.1 for DDPG to couple exploration noise that is used for the DDPG algorithm and target distribution of the KL Divergence. For TD3,

we assign σ to 0.2 since TD3 uses SARSA like updates on bootstrapping step of the algorithm by adding a noise term to the action produced by the Actor Target network. This noise component is sampled from a Gaussian Distribution with a 0.2 variance.

We use Proportional PER for the comparison. The alpha and the beta parameters set to 0.6 and 0.4 for Proportional PER [25].

For the comparison of the methods, we run the algorithms on 6 different MuJoCo tasks as the learning environments , which vary in terms of state and action spaces. We evaluate the performances of the algorithms for every 5,000 timesteps. During the evaluation episodes, agents perform by using their most recent policy 5 times. We assign average cumulative reward over 5 evaluation episodes as the evaluation score of an agent.

4.2.2 Results for DDPG

In this section, we combine KLPER, PER and Vanilla ER with the DDPG algorithm. In Figures 4.1, 4.3, 4.5, 4.7, 4.9, and 4.11, we propose the learning curves.

Figures 4.1, 4.3, 4.5, 4.7, 4.9, and 4.11 show that our method outperforms Vanilla ER and PER methods for four of the six learning environments in terms of the model's final performance and sample efficiency. Cumulative Rewards of the DDPG Agents that use KLPER as the experience replay method almost monotonically increase during the training process for most of the learning environments. For the LunarLanderContinuous-v2, Vanilla ER and KLPER converge to a nearly optimal policy at the early stages of the training. Results on InvertedPendulum-v2 suggest that there is no significant difference among the methods.

We compare each Batch Generating Policy's deviation from the most recent policy of the agent for each method and provide the results in Figures 4.2, 4.4, 4.6, 4.8, 4.10, and 4.12. As an interesting finding, even though KLPER samples 4 batches and selects the batch that has the least KL score, mostly batches that selected by PER have lower KL scores during the training process. As an explanation for that, we zoom in on the dynamics of the experience replay. The agent fills its replay buffer after the exploration steps by using its behavior policy which is a stochastic variant of its target policy. If the policy of the agent changes considerably after each policy update, then more off policy transitions would be stored to the replay buffer. We conjecture that KLPER leads more prominent policy changes during the training. Consequently, the sampled batches' KL scores are relatively high when compared to PER.

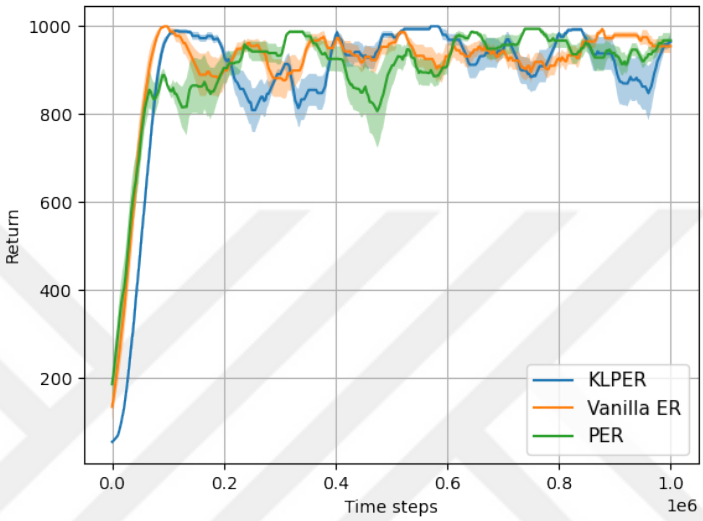

Figure 4.1: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on InvertedPendulum-v2 task. The algorithms are coupled with the DDPG.

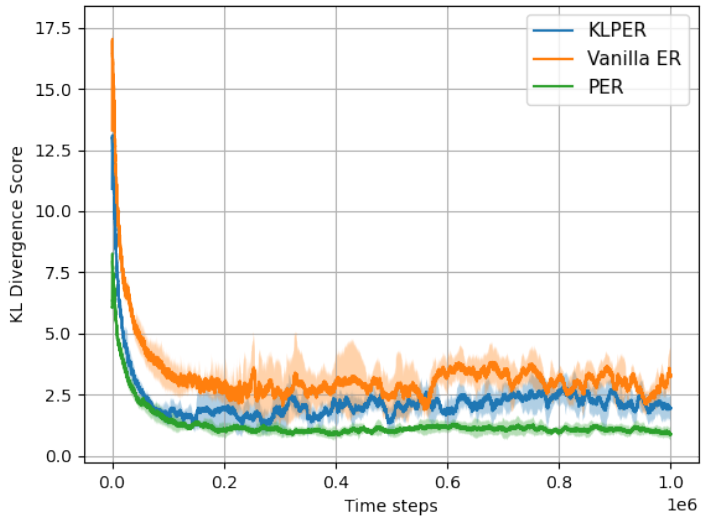

Figure 4.2: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on InvertedPendulum-v2 task.

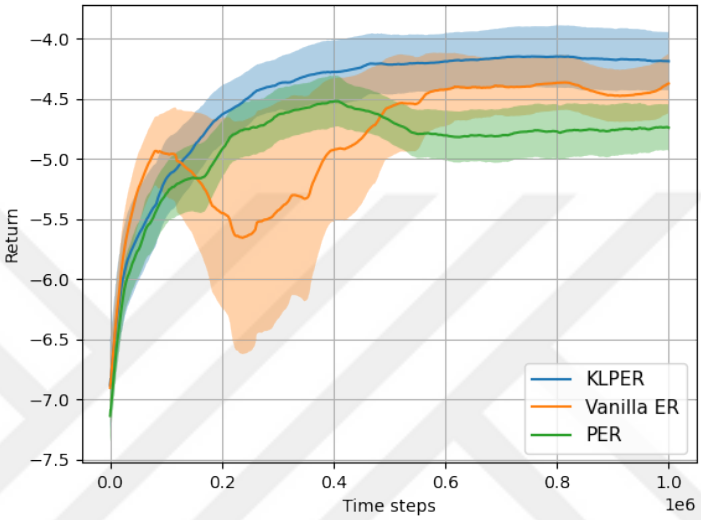

Figure 4.3: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Reacher-v2 task. The algorithms are coupled with the DDPG.

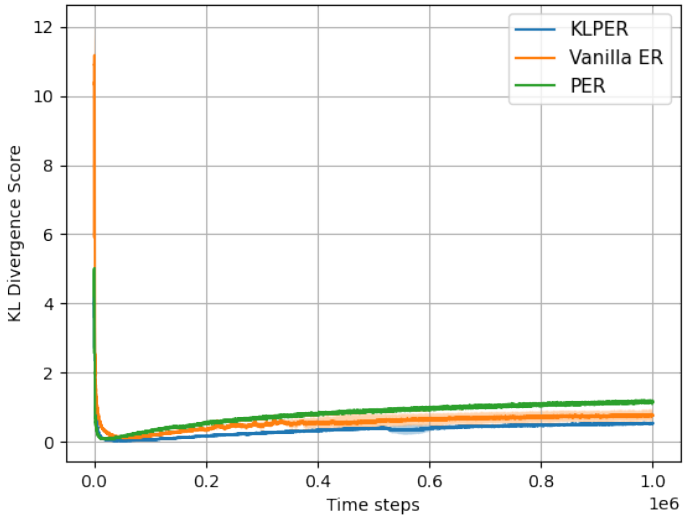

Figure 4.4: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on Reacher-v2 task.

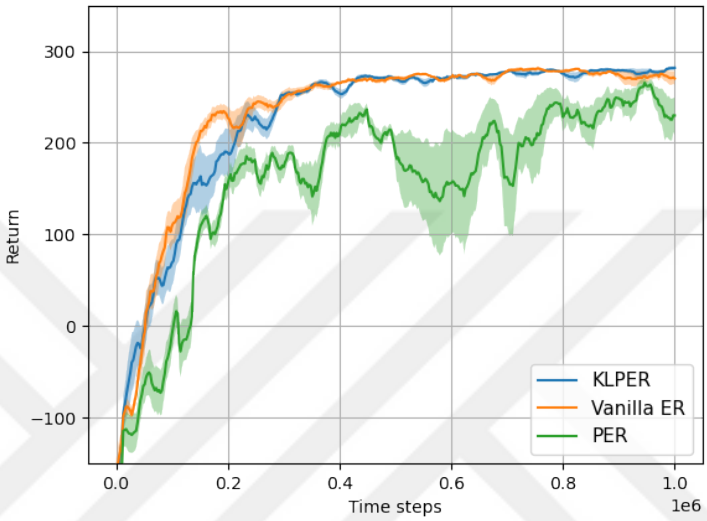

Figure 4.5: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on LunarLanderContinuous-v2 task. The algorithms are coupled with the DDPG.

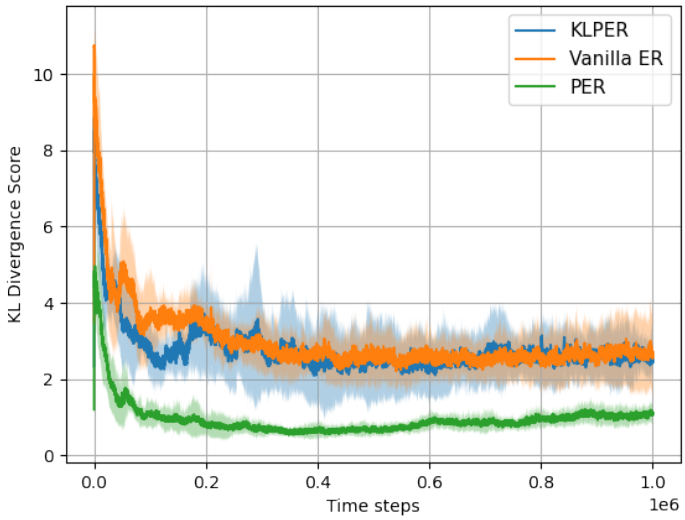

Figure 4.6: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on LunarLanderContinuous-v2 task.

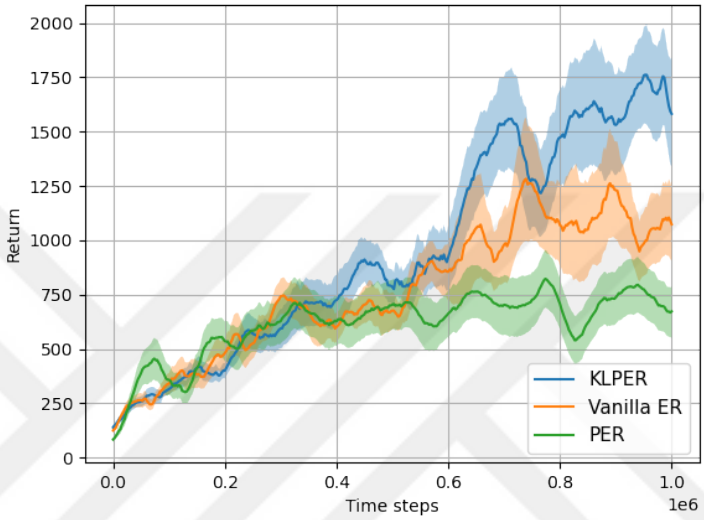

Figure 4.7: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Hopper-v2 task. The algorithms are coupled with the DDPG.

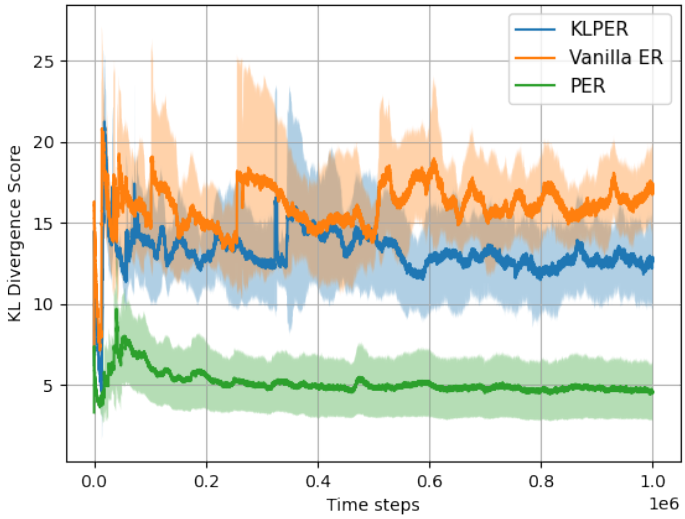

Figure 4.8: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on Hopper-v2 task.

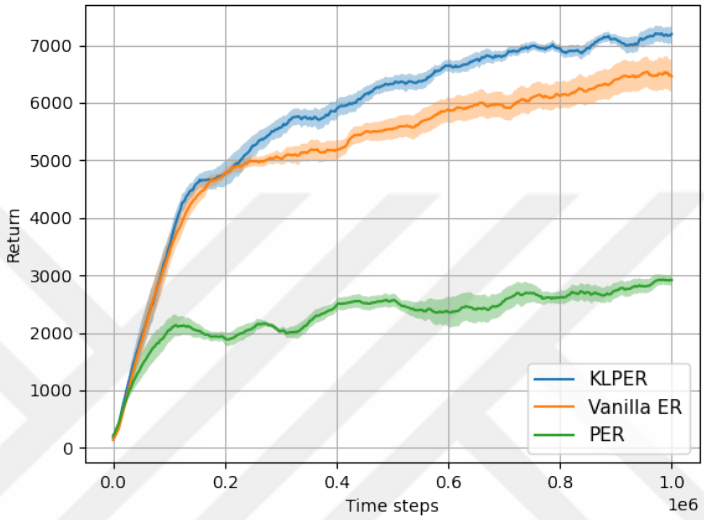

Figure 4.9: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on HalfCheetah-v2 task. The algorithms are coupled with the DDPG.

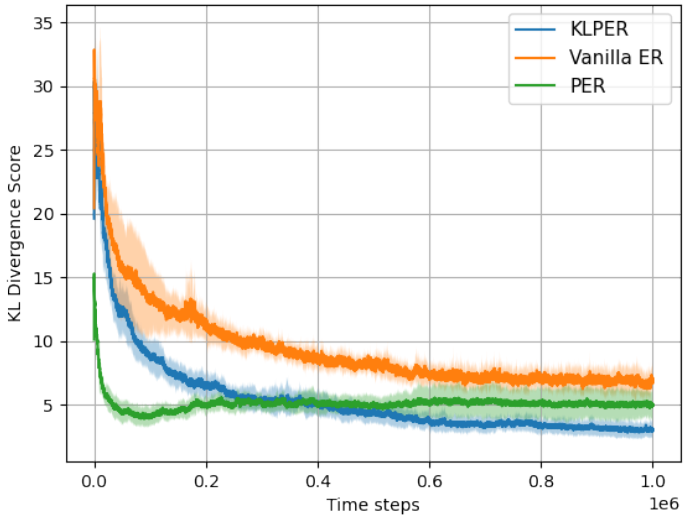

Figure 4.10: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on HalfCheetah-v2 task.

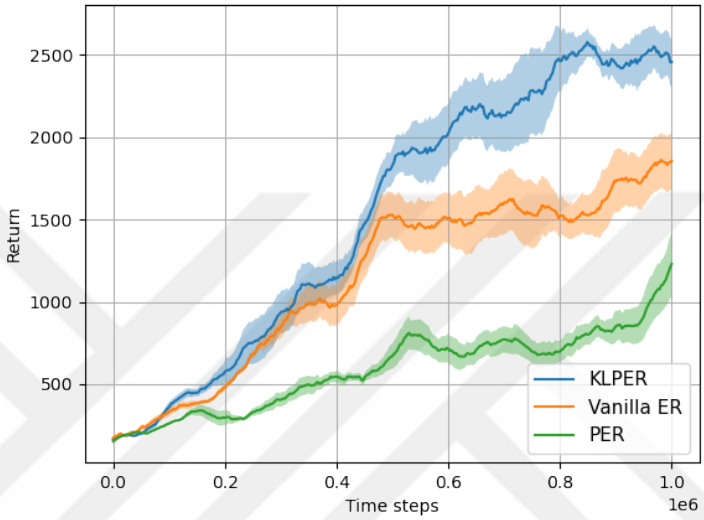

Figure 4.11: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Walker2d-v2 task. The algorithms are coupled with the DDPG.

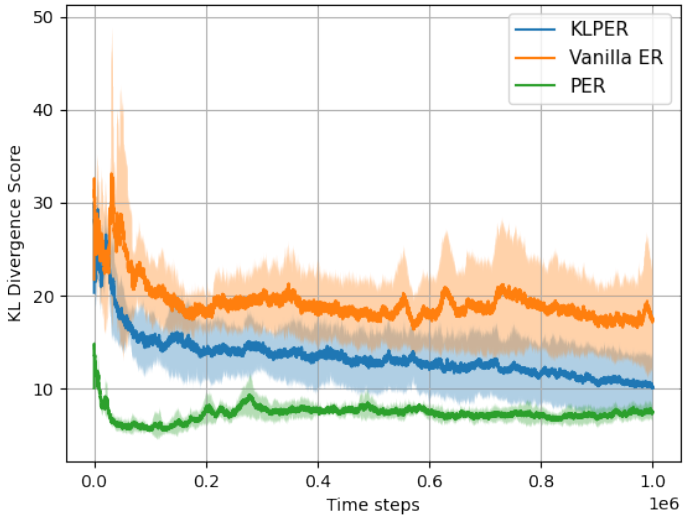

Figure 4.12: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the DDPG on Walker2d-v2 task

4.2.3 Results for TD3

In this section, we combine KLPER, PER, and Vanilla ER with the TD3 algorithm. In Figures 4.13, 4.15, 4.17, 4.19, 4.21, and 4.23, we propose learning curves.

KLPER outperforms the agents that use Vanilla ER and PER in four out of six learning environments in terms of the final performance and sample efficiency. For Reacher-v2, agent performances are almost indistinguishable. For LunarLanderContinuous-v2, the variances of the cumulative rewards collected by the agents that use Vanilla ER and PER are relatively high. On the other hand, all the agents that use KLPER converged to a nearly optimal and robust policy. KLPER provides the same performance as the other methods for the InvertedPendulum-v2 task when coupled with both the learning algorithms, DDPG and TD3. The InvertedPendulum-v2 task has four-dimensional state space and one-dimensional action space. Therefore, we explain that fact as KLPER algorithm may fail to work in low dimensional tasks. However, it works well on high-dimensional action and state spaces.

We investigate KL scores of the sampled batches for each method by following the same procedure as in DDPG and provide the results in In Figures 4.14, 4.16, 4.18, 4.20, 4.22, and 4.24. It is surprising that KL Divergence values of batches when PER and KLPER are used are so close to each other in LunarLanderContinuous-v2 and Hopper-v2. However, KLPER agents perform better than PER in both environments in terms of the final performance. Prioritizing experience replay leads to relatively smaller policy changes on these learning tasks. For InvertedPendulum-v2, KL score curves are noisy for each method. Then we observe that all the experience replay methods, when combined with TD3, do not ensure a robust policy for InvertedPendulum-v2 task.

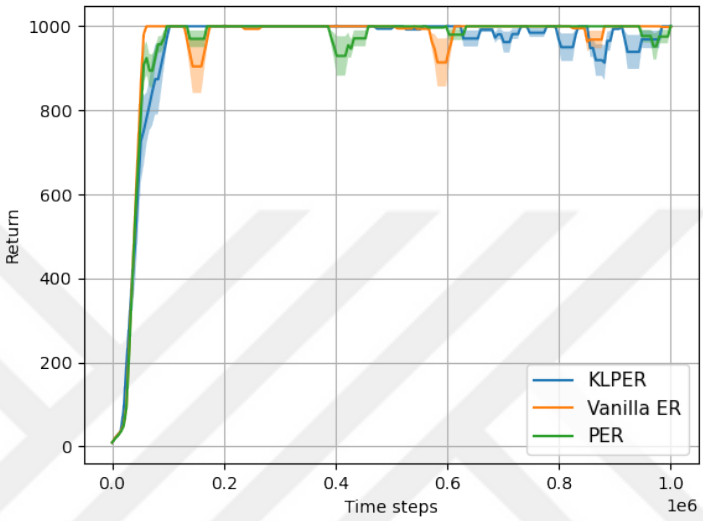

Figure 4.13: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on InvertedPendulum-v2 task. The algorithms are coupled with the TD3.

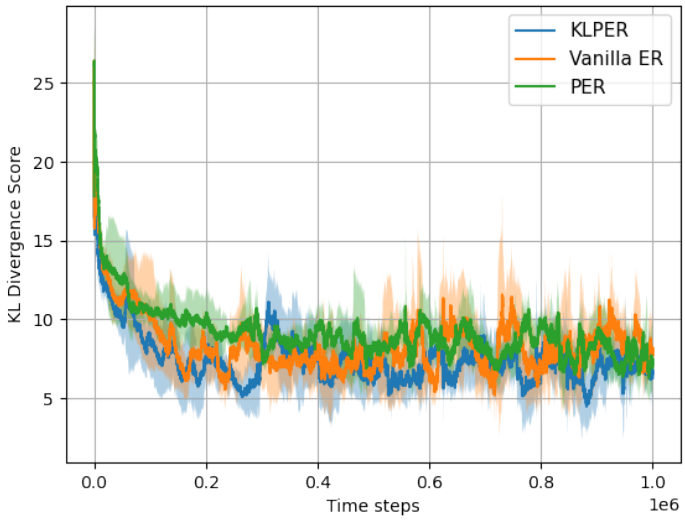

Figure 4.14: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on InvertedPendulum-v2 task.

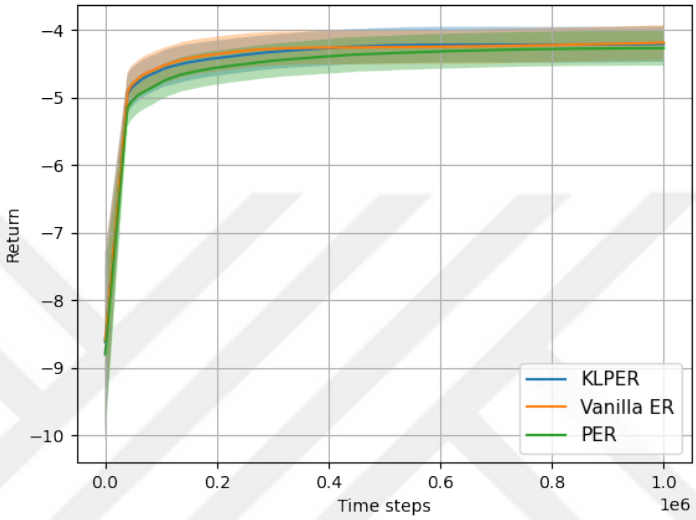

Figure 4.15: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Reacher-v2 task. The algorithms are coupled with the TD3.

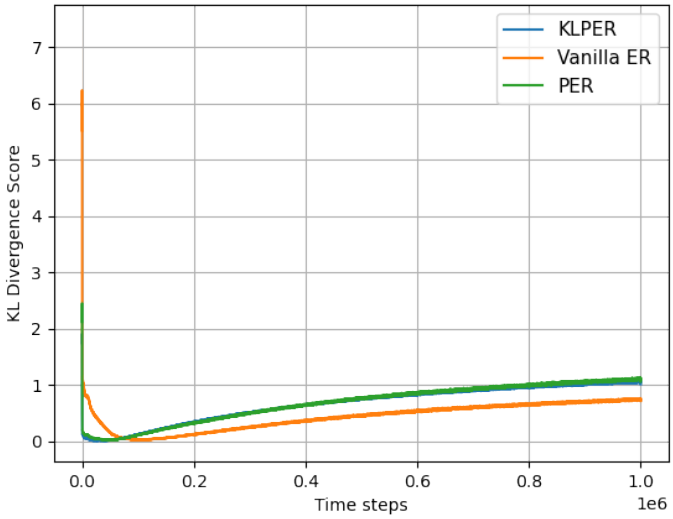

Figure 4.16: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on Reacher-v2 task.

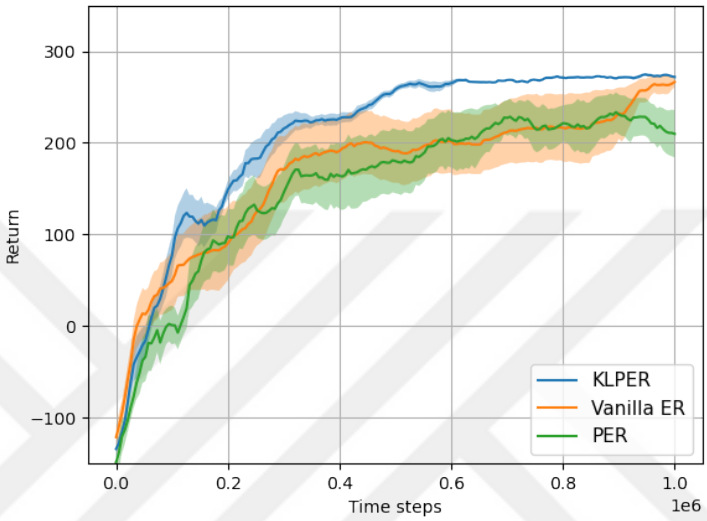

Figure 4.17: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on LunarLanderContinuous-v2 task. The algorithms are coupled with the TD3.

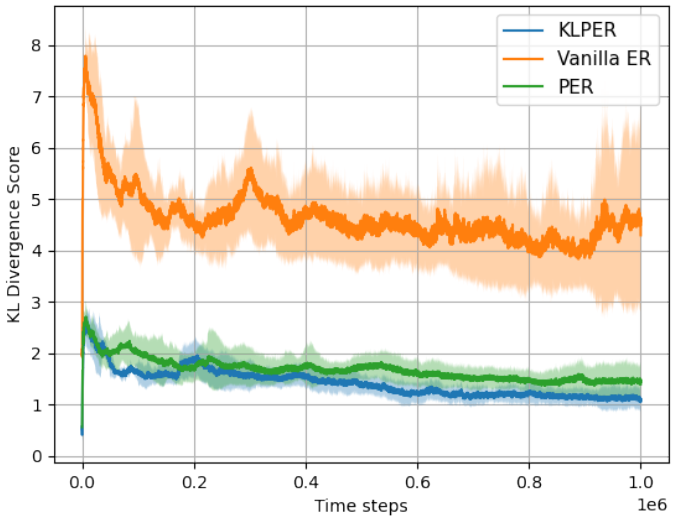

Figure 4.18: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on LunarLanderContinuous-v2 task.

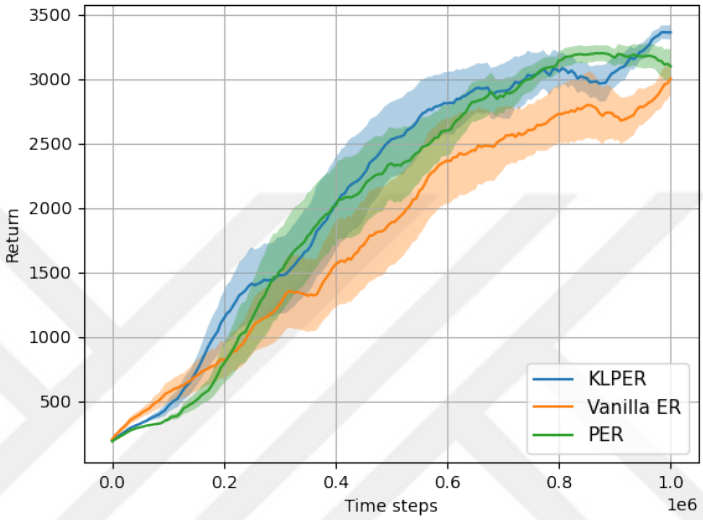

Figure 4.19: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Hopper-v2 task. The algorithms are coupled with the TD3.

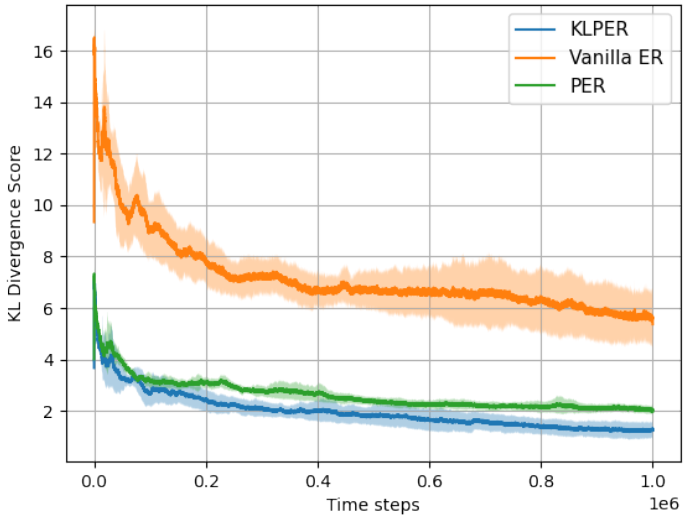

Figure 4.20: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on Hopper-v2 task.

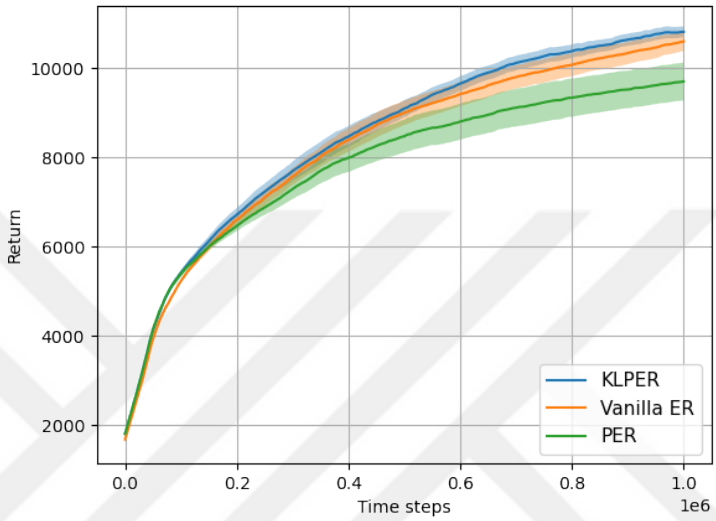

Figure 4.21: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on HalfCheetah-v2 task. The algorithms are coupled with the TD3.

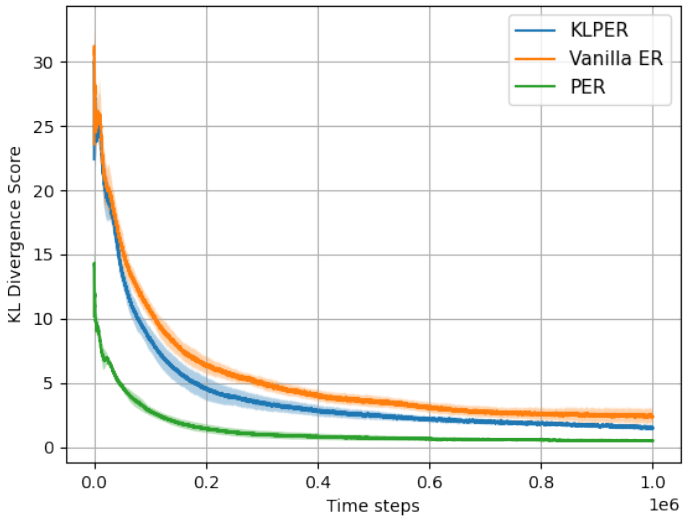

Figure 4.22: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on HalfCheetah-v2 task.

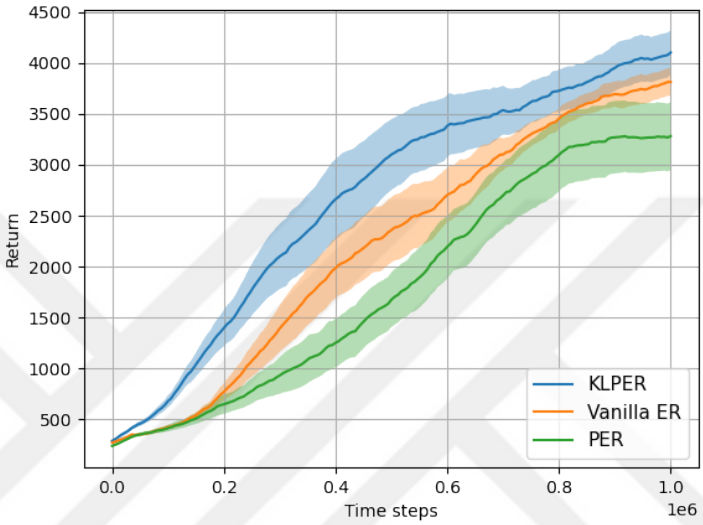

Figure 4.23: Learning Curve of the experience replay methods, KLPER, PER and Vanilla ER on Walker2d-v2 task. The algorithms are coupled with the TD3.

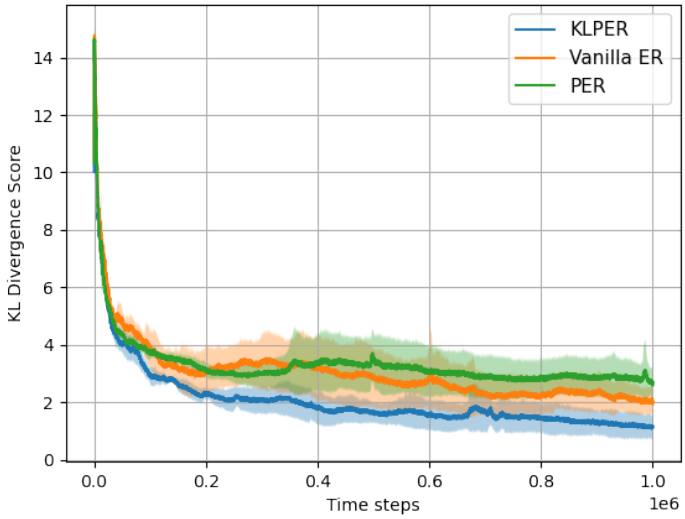

Figure 4.24: KL Divergence scores that are yielded by Batch Generating Policy and multivariate Gaussian distribution with mean 0 and covariance $0.1\mathbb{I}$ for each algorithm that coupled with the TD3 on Walker2d-v2 task

4.3 Experimental Results for DPER

4.3.1 Implementation Details

We compare our algorithms with KLPER, PER, and Vanilla ER. We run 10 trials for each algorithm, and to make a fair comparison, we use the same seeds for each algorithm. We implement PER and KLPER algorithms by following the original implementations. We use candidate batch numbers as 4 rather than 8 for KLPER to reduce the computational complexity. We use the Proportional variant of the PER for the comparison [30]. The alpha and the beta parameters set to 0.6 and 0.4 for Proportional PER [25]. We note that our algorithm, DPER, does not include the beta parameter that PER uses since both alpha and beta parameters mainly serve to anneal the bias that the algorithm introduces.

We couple DPER, KLPER, PER, and Vanilla ER with TD3. We also use its original implementation, which is given in the thesis that proposes the algorithm [33]. The TD3 has three-layered neural networks for both the Actor and the Critic networks, including two hidden layers with 256 neurons. We use Adam [40] as the optimizer for both of the primary networks, the Actor-Network and the Critic-Network. We observe that the TD3 algorithm converges to a local optimum policy when the number of exploration steps is insufficient. Therefore, we fill the replay buffer with 25,000 transitions gathered by the agent while acting randomly for the TD3 algorithm for each experience replay method.

We choose N, an adjustable parameter that represents the number of candidate batches in our algorithm that we inherit from KLPER, as 4 in our implementation. We choose σ as 0.2 for TD3 to couple exploration noise that is used for the TD3 algorithm and target distribution of the KL Divergence since TD3 uses SARSA like updates on bootstrapping step of the algorithm by adding a noise term to the action produced by the Actor-Target-Network.

We run the algorithms on 6 different MuJoCo and OpenAi gym tasks as the learning environments. These environments vary in terms of state and action

spaces. We evaluate the performances of the algorithms for every 5,000 timesteps. During the evaluation episodes, agents perform by using their most recent policy 10 times. We assign the average cumulative reward over 10 evaluation episodes as the return of an agent.

4.3.2 Results for TD3

In this section, we discuss the metrics that we collected during the training and represent the learning curves of DPER, KLPER, PER, and Vanilla ER when combined with the TD3 algorithm.

We illustrate the Average Magnitude of the Temporal Difference Error that batch of transitions used for the Critic-Network training at each timestep in Figures 4.31, 4.32, 4.33, 4.34, 4.35, and 4.36. It is apparent that our algorithm and PER choose transitions that have higher magnitudes of Temporal Difference Error for every learning task.

We depict the KL divergence value yielded by the batch of the transitions that are used for the Actor-Network training for each timestep in Figures 4.25, 4.26, 4.27, 4.28, 4.29, and 4.30. For all learning tasks, the Actor-Network starts to learn through more off-policy transitions caused by the 25,000 exploration steps. It is confounding that even though our algorithm forces the agent to use more on-policy transitions, this is clearly not the case for the BipedalWalker-v3 environment. On the other hand, our algorithm is effective in terms of reducing the off-policiness of the learning algorithm for the Ant-v2 and HalfCheetah-v2 environments.

We emphasize that reducing off-policiness of an algorithm should be inspected carefully. If one component of the experience replay method causes significant changes on-policy of the agent during the training, it may not be possible to reach desired on-policiness level. This is the case that we encounter for the BipedalWalker-v3 environment. Our algorithm is significantly outperformed by the KLPER and Vanilla Er, which both use more on-policy transitions in this

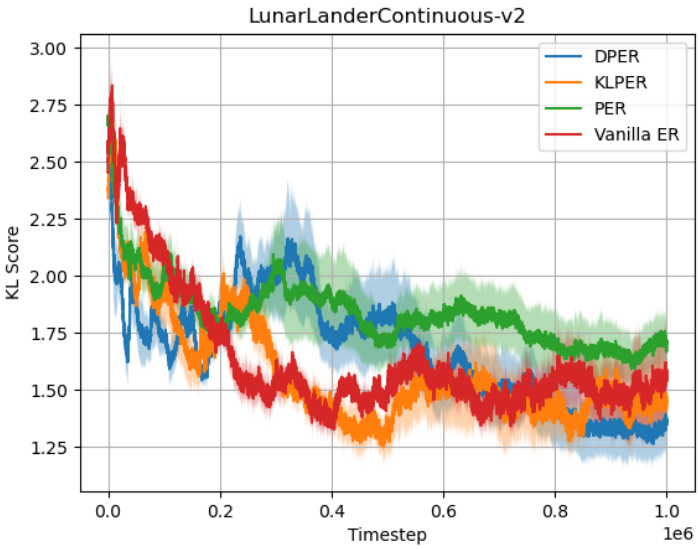

Figure 4.25: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on LunarLanderContinuous-v2 task. The algorithms are coupled with the TD3.

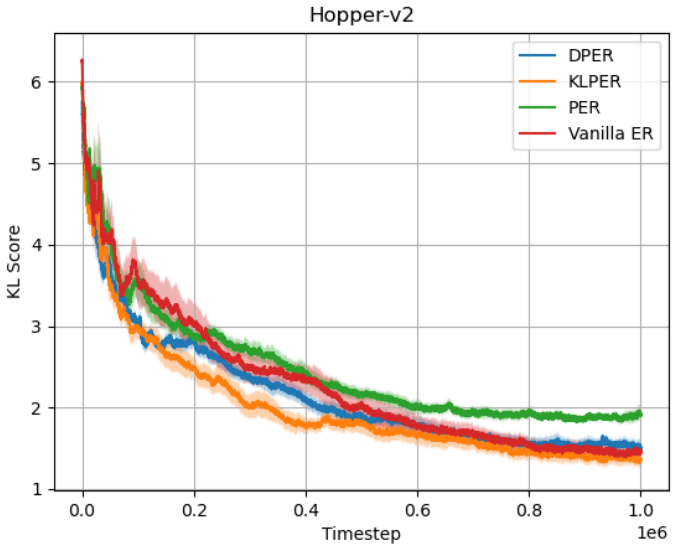

Figure 4.26: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Hopper-v2 task. The algorithms are coupled with the TD3.

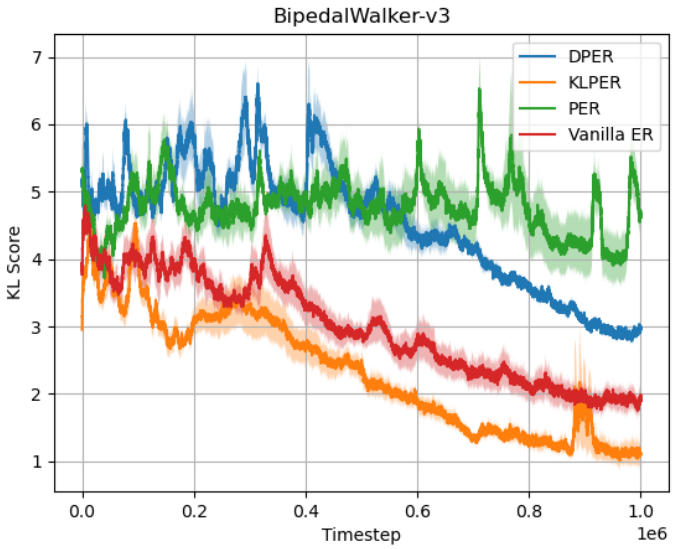

Figure 4.27: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on BipedalWalker-v3 task. The algorithms are coupled with the TD3.

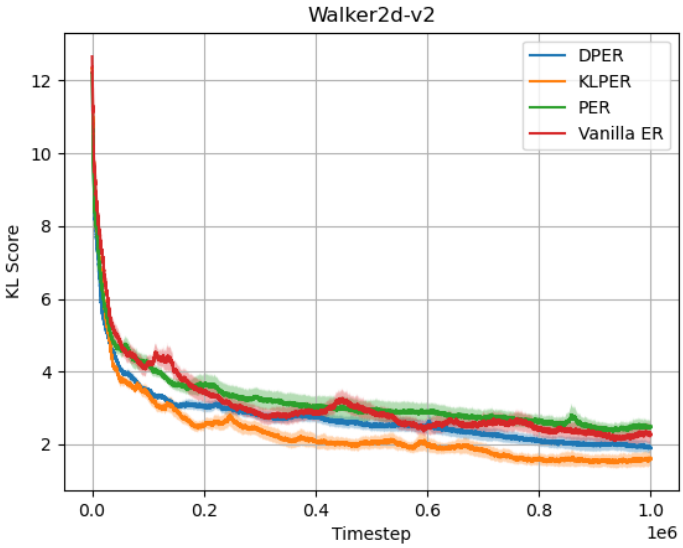

Figure 4.28: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Walker2d-v2 task. The algorithms are coupled with the TD3.

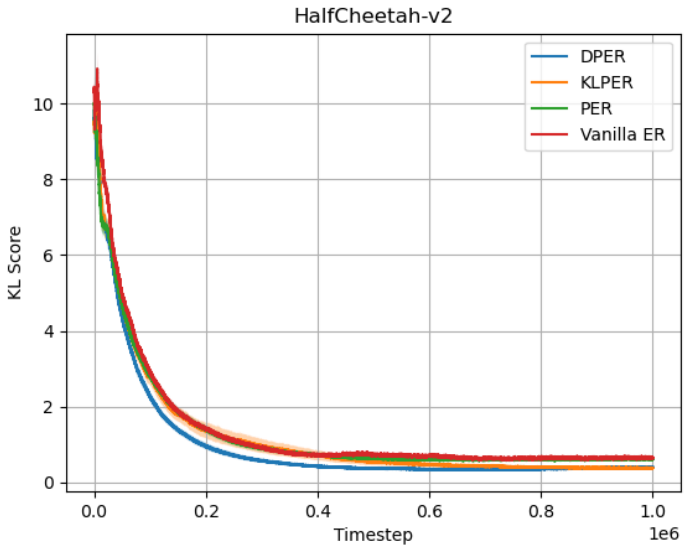

Figure 4.29: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on HalfCheetah-v2 task. The algorithms are coupled with the TD3.

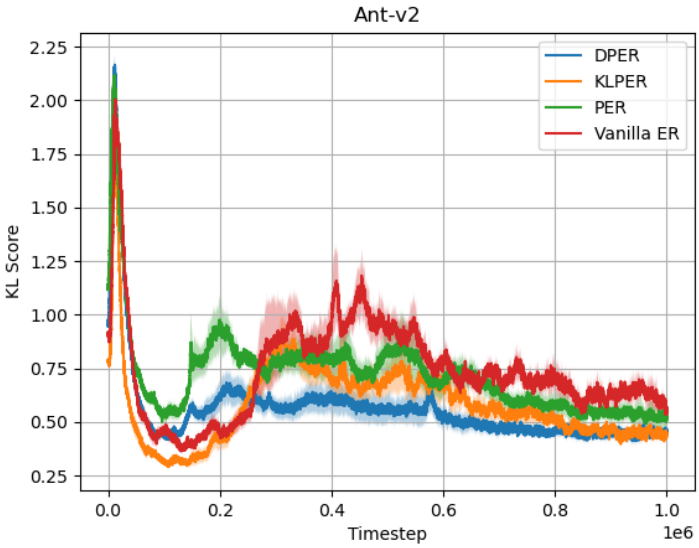

Figure 4.30: KL scores that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Ant-v2 task. The algorithms are coupled with the TD3.

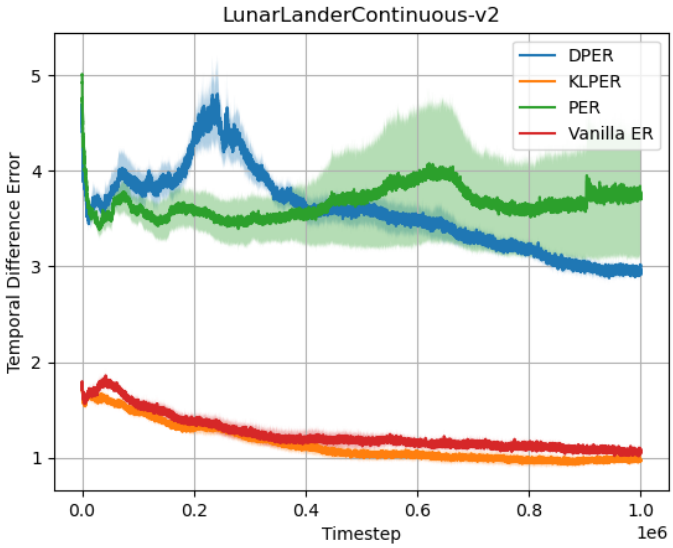

Figure 4.31: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on LunarLanderContinuous-v2. The algorithms are coupled with the TD3.

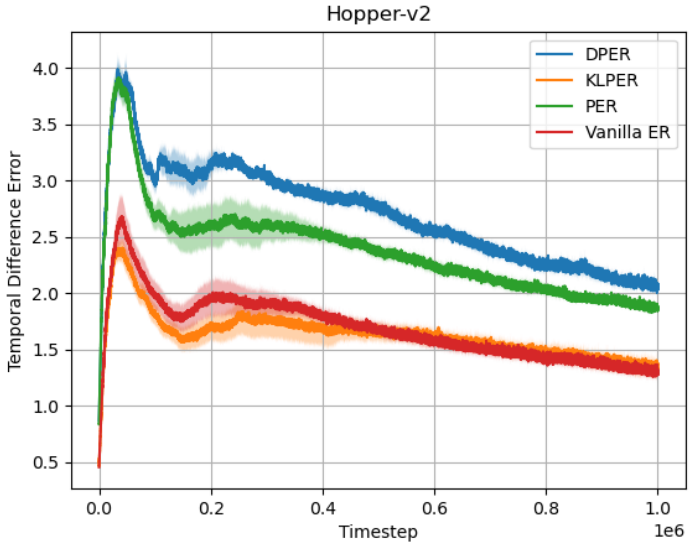

Figure 4.32: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Hopper-v2. The algorithms are coupled with the TD3.

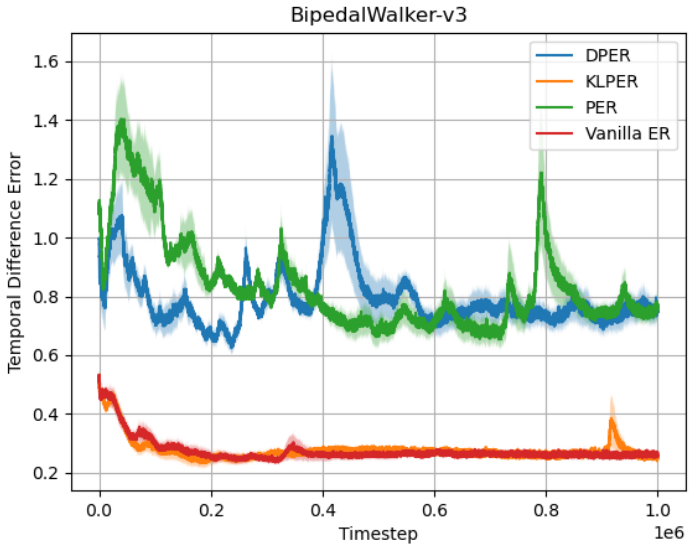

Figure 4.33: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on BipedalWalker-v3. The algorithms are coupled with the TD3.

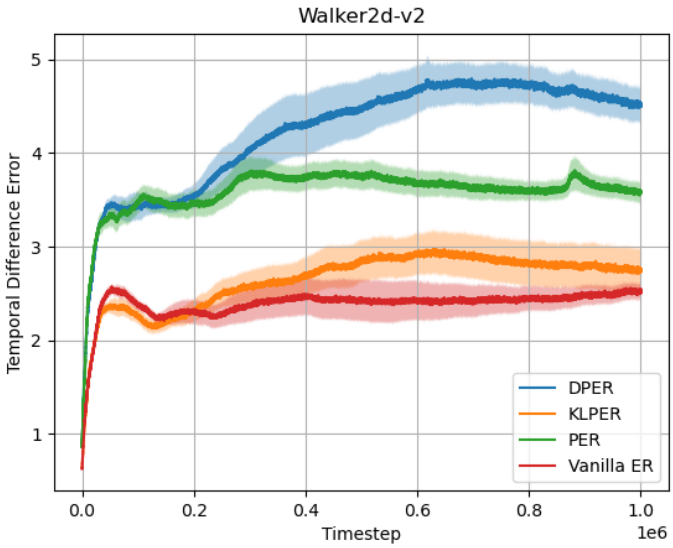

Figure 4.34: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Walker2d-v2. The algorithms are coupled with the TD3.

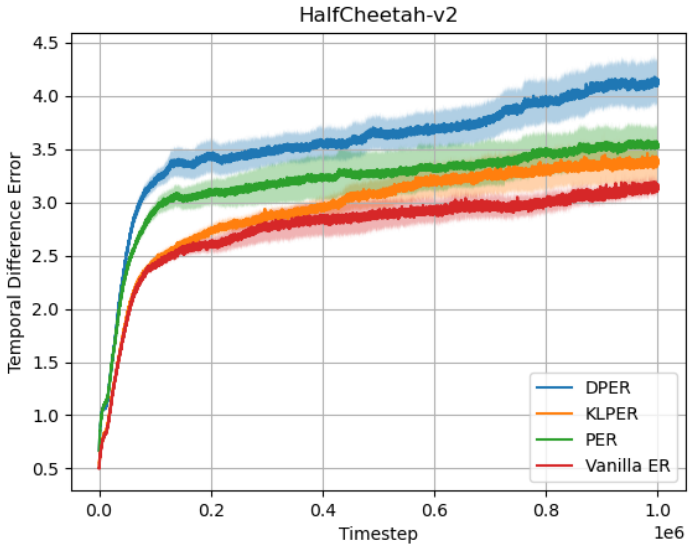

Figure 4.35: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on HalfCheetah-v2. The algorithms are coupled with the TD3.

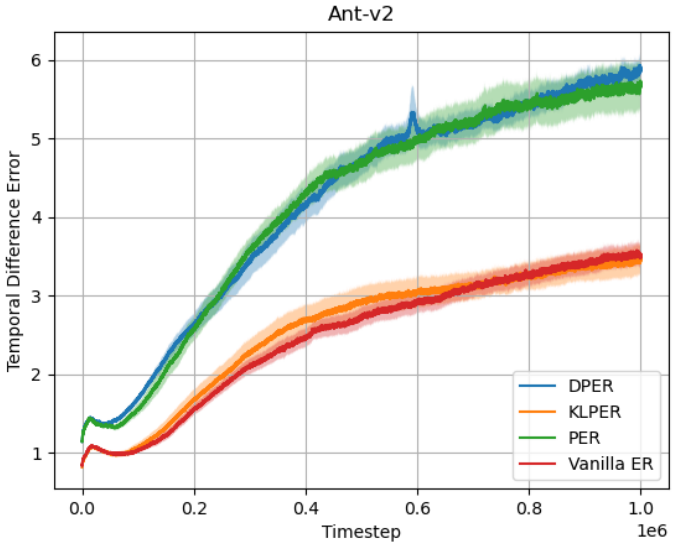

Figure 4.36: The average magnitude of the temporal difference error that batch of transitions yield each timesteps for DPER, KLPER, PER and Vanilla ER on Ant-v2. The algorithms are coupled with the TD3.

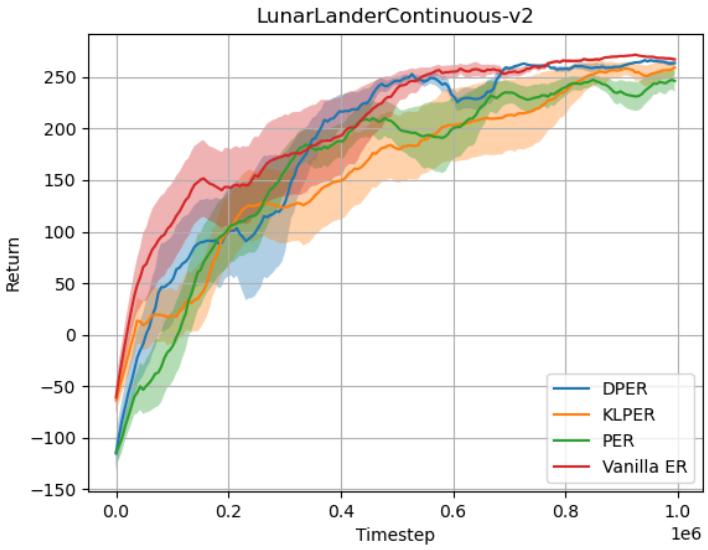

Figure 4.37: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on LunarLanderContinuous-v2. The algorithms are coupled with the TD3.

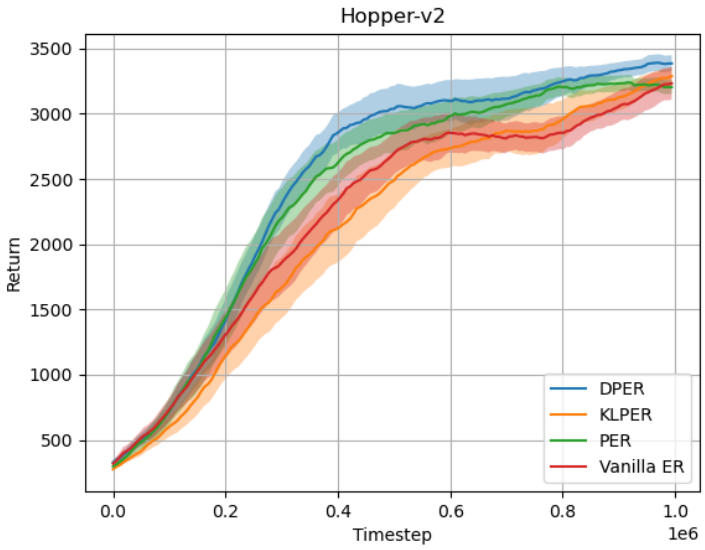

Figure 4.38: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on Hopper-v2. The algorithms are coupled with the TD3.

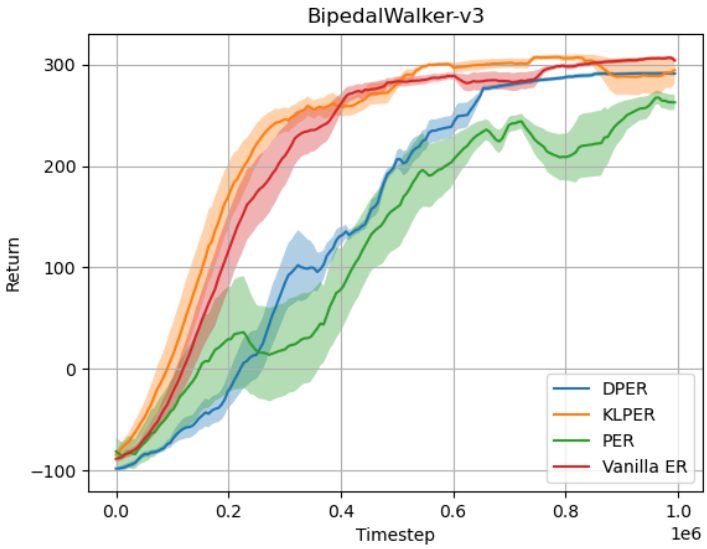

Figure 4.39: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on BipedalWalker-v3. The algorithms are coupled with the TD3.

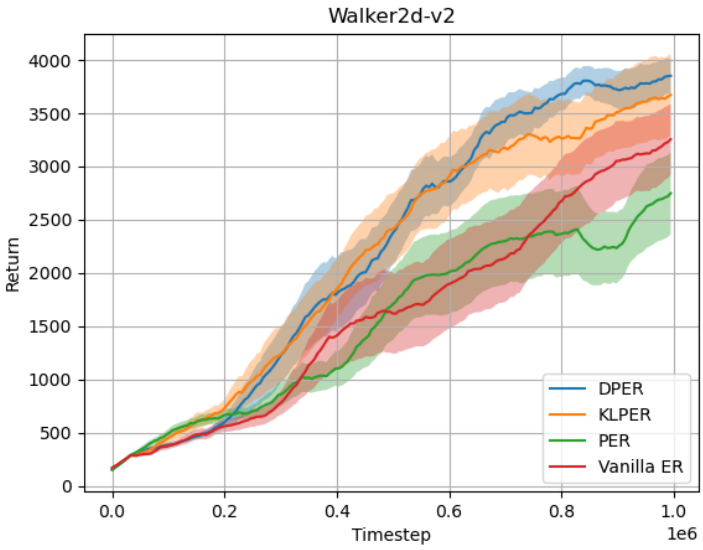

Figure 4.40: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on Walker2d-v2. The algorithms are coupled with the TD3.

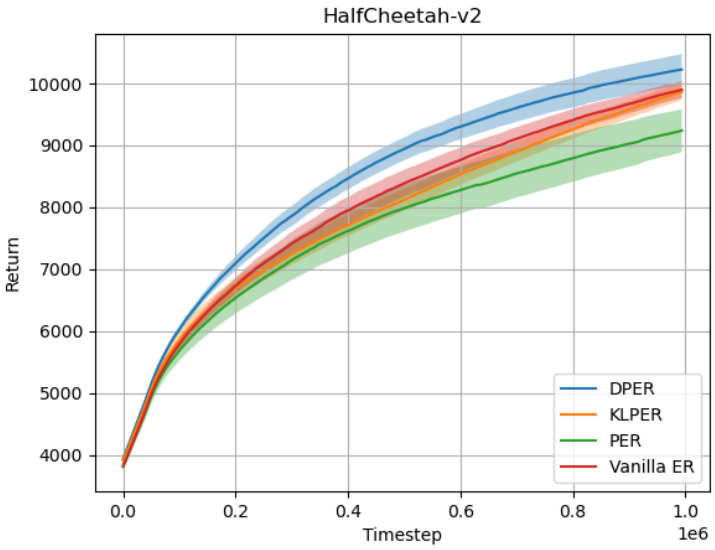

Figure 4.41: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on HalfCheetah-v2. The algorithms are coupled with the TD3.

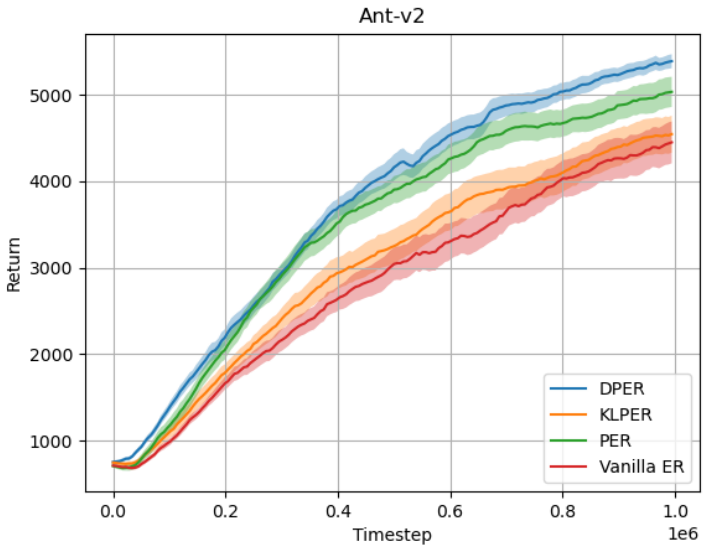

Figure 4.42: Learning Curves of the experience replay methods, DPER, KLPER, PER and Vanilla ER on Ant-v2. The algorithms are coupled with the TD3.

learning task. We highlight that our agent barely accomplished using more on-policy batches of transitions and transitions that output higher magnitudes of Temporal Difference Error in Ant-v2 and HalfCheetah-v2 environments. Moreover, our algorithm significantly outperforms other algorithms in these two environments.

Overall, our algorithm, DPER, outperforms the agents that use KLPER, PER, and Vanilla ER in four out of six learning environments in terms of the final performance and sample efficiency. In BipedalWalker-v3 and LunarLanderContinuous-v2 environments, our algorithm is outperformed by the other methods. In these environments, we observe dominant and unexpected policy changes. The spikes in Figures 4.27, 4.30 represent these changes. We claim that significant policy changes may affect the performances of the agents negatively for all of the experience replay methods we use in this work and should be inspected carefully.

Chapter 5

Conclusion

In this thesis, we emphasize the drawbacks of prioritizing transitions and improving the performance of the agent by reducing the off-policiness of the reinforcement learning algorithms. We introduce the KLPER algorithm that makes prioritization the batch of transitions rather than prioritizing each transition in the replay buffer. We define Batch Generating Policy to quantize the off-policiness level of a batch. Our algorithm enables agents to have more on-policy updates using KL Divergence between Batch Generating Policy and a multivariate Gaussian distribution with a mean of 0. We combine KLPER with Deep Deterministic Policy Gradient algorithms and test it on continuous control tasks. Results show that KLPER brings promising improvements and outperforms Vanilla ER and PER in terms of sample efficiency and final performance in particular continuous control tasks.

We also propose a method that addresses the drawbacks of other conventional experience replay mechanisms. We improve two mutually exclusive experience replay methods by using them on the Actor-Network and the Critic-Network training separately. We introduce DPER, which decouples the training principles of the Actor-Network and the Critic-Network. Results show that DPER presents promising improvements and outperforms KLPER, PER, and Vanilla ER in terms of sample efficiency and final performance in particular continuous control tasks.

Also, we state that the average magnitude of the temporal difference error and KL scores yielded by the batch of transitions that are used during the training have crucial importance for the learning process. Therefore, we claim that the properties of the batch of transitions that are used for the training for both the Actor and the Critic should be sifted through and may lead to more studies.

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," *arXiv preprint arXiv:1509.02971*, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den 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, jan 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] A. Ghadirzadeh, X. Chen, W. Yin, Z. Yi, M. Björkman, and D. Kragic, "Human-centered collaborative robots with deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 566–571, 2021.

[6] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu, "High-speed autonomous drifting with deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1247–1254, 2020.

[7] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[8] L. Guzman, V. Morellas, and N. Papanikolopoulos, "Robotic embodiment of human-like motor skills via reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3711–3717, 2022.

[9] M. Yuan, J. Shan, and K. Mi, "Deep reinforcement learning based game-theoretic decision-making for autonomous vehicles," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 818–825, 2022.

[10] M. Turan, Y. Almalioglu, H. B. Gilbert, F. Mahmood, N. J. Durr, H. Araujo, A. E. Sarı, A. Ajay, and M. Sitti, "Learning to navigate endoscopic capsule robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3075–3082, 2019.

[11] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, p. 293–321, May 1992.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] 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.

[17] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[18] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016.

[19] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, no. 9, pp. 1–56, 2018.

[20] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[21] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[22] G. Novati and P. Koumoutsakos, "Remember and forget for experience replay," in *International Conference on Machine Learning*, pp. 4851–4860, PMLR, 2019.

[23] H. Liu, A. Trott, R. Socher, and C. Xiong, "Competitive experience replay," *arXiv preprint arXiv:1902.00528*, 2019.

[24] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.

[25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[26] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to sample with local and global contexts in experience replay buffer," *arXiv preprint arXiv:2007.07358*, 2020.

[27] P. Sun, W. Zhou, and H. Li, "Attentive experience replay," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5900–5907, Apr. 2020.

[28] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Experience replay optimization," *arXiv preprint arXiv:1906.08387*, 2019.

[29] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

[30] 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)*, pp. 1255–1262, 2021.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

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

[33] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[34] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[35] 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.

[36] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[37] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Review*, vol. 36, pp. 823–841, Sept. 1930.

[38] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," 2019.

[39] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," *CoRR*, vol. abs/1604.06778, 2016.

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