

RESEARCH ARTICLE

MP-TD3: Multi-Pool Prioritized Experience Replay-Based Asynchronous Twin Delayed Deep Deterministic Policy Gradient Algorithm

WENWEN TAN¹ AND DETIAN HUANG^{1,2}, (Member, IEEE)

¹College of Engineering, Huaqiao University, Quanzhou 362000, China

²Quanzhou Digital Institute, Quanzhou 362000, China

Corresponding author: Detian Huang (huangdetian@hqu.edu.cn)


This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFE0205400, in part by the National Natural Science Foundation of China under Grant 61901183 and Grant 61976098, in part by the Fundamental Research Funds for the Central Universities under Grant ZQN-921, in part by the Collaborative Innovation Platform Project of Fujian Province under Grant 2021FX03, in part by the Natural Science Foundation of Fujian Province under Grant 2023J01140, in part by the Natural Science Foundation of Fujian Provincial Science and Technology Department under Grant 2021H6037, in part by the Key Project of Quanzhou Science and Technology Plan under Grant 2021C008R and Grant 2023C007R, and in part by the Key Science and Technology Project of Xiamen City under Grant 3502Z20231005.

ABSTRACT The prioritized experience replay mechanisms have achieved remarkable success in accelerating the convergence of reinforcement learning algorithms. However, applying traditional prioritized experience replay mechanisms directly to asynchronous reinforcement learning leads to slow convergence, due to the difficulty for an agent to utilize excellent experiences obtained by other agents interacting with the environment. To address the above issue, we propose a Multi-pool Prioritized experience replay-based asynchronous Twin Delayed Deep Deterministic policy gradient algorithm (MP-TD3). Specifically, a multi-pool prioritized experience replay mechanism is proposed to strengthen the experience interactions among different agents to accelerate the network convergence. Then, a global-pool self-cleaning mechanism based on sample diversity and a global-pool self-cleaning mechanism based on TD-errors are designed to overcome the deficiency that the samples suffer from high redundancy and low information content in the global-pool, respectively. Finally, a multi-batch sampling mechanism is investigated to further reduce the training time. Extensive experiments validate that the proposed MP-TD3 significantly improve the convergence speed and performance compared with state-of-the-art methods.

INDEX TERMS Asynchronous reinforcement learning, twin delayed deep deterministic algorithm, prioritized experience replay, TD-error.

I. INTRODUCTION

Reinforcement Learning (RL) [1], [2] is one of the promising machine learning methods for obtaining optimal decisions through iterative trial and error, where an agent constantly interacts with environment to improve its strategy for maximum cumulative rewards. It has been widely used

The associate editor coordinating the review of this manuscript and approving it for publication was Deepak Mishra .

in various fields such as robot [3], [4], [5], autonomous vehicles [6], [7], supply chain optimization [8], [9] and game design [10], [11], and so on. In recent years, numerous variants of RL techniques have been developed. For example, Watkins [12] proposed a Q-learning algorithm that allows agents to learn optimal policy in a controlled Markov Decision Process (MDP) [13]. William [14] proposed a connectionist RL algorithm, which only requires learning of the policy function and not the value function. Konda and

Tsitsiklis [15] proposed an Actor-Critic (AC) framework, which estimates not only the policy function but also the value function.

In previous RL algorithms, an agent uses the experience derived from a single interaction with the environment to update the model, which means that the interaction experience is discarded after a single update. This results in two following problems: 1) the interaction experience has highly temporal correlation, which is detrimental to the model training; 2) the interaction experience that is rapidly discarded may be a rare experience that is useful for subsequent training. To solve both problems, the experience replay mechanism [16] stores the experiences obtained from the interaction of agent with environment in an experience replay pool, and then the agent is able to update the model according to the previous and recent experiences. In other words, the experience replay mechanism addresses the problem of temporal correlation between training data. On the other hand, a number of significant experiences have been used repeatedly to update the model, which effectively improves the learning efficiency of the agent.

Recently, numerous studies have revealed the advantages of introducing experience replay in RL algorithms. Deep Q-Network (DQN) algorithm [17] stabilizes the training of the value function represented by a deep neural network with an experience replay mechanism. Deep Deterministic policy gradient (DDPG) algorithm [18] also introduces an experience replay mechanism, which selects state transfer samples from the environment by an exploration strategy and delivers the samples into an experience replay pool, so as to reduce the number of interactions between the agent and the environment. Although the above RL algorithms are effective in exploring the optimal policy for the environment, they all employ uniform sampling, *i.e.*, they do not take into account the significance of the samples, resulting in all samples being selected with the same probability. A recent work by DeepMind [19] pointed out that the experience replay mechanism neglects the temporal property of the samples and the efficiency of sampling the agents [20], [21], [22], [23], [24], which is prone to result in unstable training and slow convergence [25].

To address the above problems, the prioritized experience replay mechanism [19] has received increasing attention. This mechanism prioritizes the samples in an appropriate order, which enables the agents to select more valuable samples in the experience replay pool. According to the different prioritizing methods, it can be divided into value change-based [26], [27] and TD error-based [28] prioritizing. Recently, numerous works have demonstrated that the prioritized experience replay accelerates training for RL [29], [30], [31]. Brittain et al. [32] proposed a Prioritized Sequence Experience Replay (PSER) algorithm, which exploits the relationship between successive experiences by assigning a higher sampling priority to an important experience, as well as increasing the priority of the antecedent sequence experiences that lead to the creation of important experience.

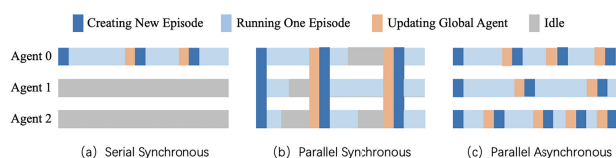


FIGURE 1. Comparison of training methods for update mechanism.

Cao et al. [33] proposed a High-Value Prioritized Experience Replay (HVP ER), which incorporates the value function with TD-error as a measure of its prioritization. Wang et al. [34] incorporated the prioritized experience replay in a dueling network.

However, application of prioritized experience replay mechanisms to asynchronous RL algorithms has yet to be further studied. Intuitively, there are two approaches to apply traditional prioritized experience replay mechanisms to asynchronous RL algorithms: 1) each agent uses its respective experience replay pool and adopts the prioritized experience replay mechanism within the experience replay pool; 2) each agent uses the same global-pool and employs the prioritized experience replay mechanism in the global-pool. The former prevents each agent from sharing its respective excellence experiences with each other, resulting in slow training speed. On the contrary, the latter makes each agent require frequent experience interactions, inevitably increasing the time overhead. As a result, it is imprudent to directly apply traditional prioritized experience replay to asynchronous RL.

To solve the above issues, a multi-pool prioritized experience replay mechanism is designed and further incorporated into the TD3 algorithm, thus forming a Multi-pool Priority experience replay-based asynchronous Twin Delayed Deep Deterministic policy gradient algorithm (MP-TD3). Experimental results that the proposed multi-pool prioritized experience replay mechanism significantly accelerates the convergence while boosting the performance of the TD3 algorithm. Specifically, the proposed MP-TD3 employs a multi-pool structure, in which various agents share the learned outstanding experiences. Then, different from existing asynchronous RL [35], our MP-TD3 adopts the proposed global-pool self-cleaning mechanisms based on sample diversity and TD-errors, which not only reduces the similarity of the samples, but also improves the information content of the samples. Next, our MP-TD3 uses the proposed self-updating mechanism based on TD-errors to alleviate the TD-error hysteresis of samples. Subsequently, a multi-batch sampling mechanism and a multi-cache structure are implemented to decrease the training time. In summary, the primary contributions of our work are as follows:

- To address the problem that prioritized experience replay mechanism suffers from the experience interactions among agents in asynchronous RL, a Multi-pool Prioritized experience replay-based asynchronous Twin Delayed Deep Deterministic policy gradient algorithm (MP-TD3) is proposed. Experimental results

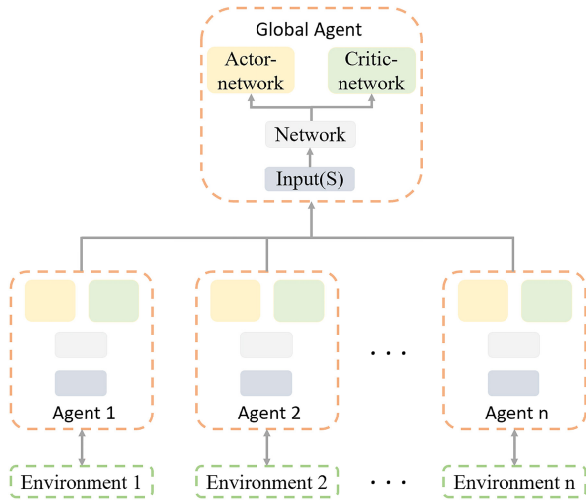


FIGURE 2. The network structure of A3C, contains an Actor-network and a Critic-network, which operate independently and in parallel.

demonstrate that the proposed MP-TD3 improves the convergence speed and performance effectively.

- A multi-pool prioritized experience replay mechanism is proposed to strengthen the experience interactions among different agents, so as to accelerate the convergence.
- Two global-pool self-cleaning mechanisms based on sample diversity and TD-errors are respectively designed to remove redundant samples and maintain the diversity of samples.
- A self-updating mechanism based on TD-errors is proposed to alleviate the TD-error hysteresis.

This paper consists of five sections. Section II discusses the basic concepts of TD3 algorithm, asynchronous RL, and prioritized experience replay mechanism based on random sampling. The detail of the proposed MP-TD3 is discussed in Section III. Section IV presents experimental results and analysis. Finally, Section V summarizes the paper and discusses future research directions.

II. PRELIMINARIES

A. TD3 ALGORITHM

Traditional RL algorithms generally employ a single-agent training approach that updates serially during training, as shown in Fig. 1(a). At each time-step t , the agent acquires the state s_t of the environment and selects an action a_t by a policy π . Upon executing the action a_t , the agent obtains a reward r_t and transitions to the next state s_{t+1} . This iterative process is repeated until the agent reaches the terminal state within the state space S .

During the interaction between the agent and environment, the discounted cumulative rewards at time-step t is defined as follows:

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

where T indicates the maximum time-step, γ is a discount factor with a range of (0,1].

To realize the ultimate goal of RL algorithm, which explores the optimal policy for maximizing the cumulative rewards. Define the action-value function $Q^\pi(s, a) = \mathbb{E}_\pi [R_t | s_t = s, a_t = a]$ to represent the expectation of the discounted cumulative rewards for an agent executing action a_t under policy π and state s . The action-value function is known as the Q-function, adheres to the Bellman equation, which is expressed as follows:

$$Q^\pi(s, a) = \mathbb{E}_{a_{t+1} \sim \pi, s_{t+1} \sim \rho^\pi} [r(s, a) + \gamma Q^\pi(s_{t+1}, a_{t+1})], \quad (2)$$

where ρ^π is the distribution of state transfer probability with policy π .

Fujimoto et al. [36] proposed a Twin Delayed Deep Deterministic policy gradient (TD3) algorithm, which mitigates network overestimation by adopting a minimum value between two Critic-networks. In addition, the algorithm delays the policy update while incorporating noise into the actions to minimize the error and improve the performance.

At each time-step, the agent selects an action $a_t = \pi_\phi(s_t) + N_t$ augmented with noise in state s , where N_t is the added noise. Then, the agent performs the action and stores the experience (s_t, a_t, s_{t+1}, r_t) obtained from interaction with environment in the experience replay pool. Subsequently, a mini-batch of transitions is selected randomly from the experience replay pool, and the two Critic-networks are updated by optimizing the loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')} [(Q_{\theta_i}(s, a) - y)^2], \quad (3)$$

$$y = r(s, a) + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi'}(s')). \quad (4)$$

In Eq. (4), TD3 calculates the values of the two Critic-networks $Q_{\theta'_i}(s', \pi_{\phi'}(s'))$ separately, and selects the smaller as the Q-value y for the Target-network. Here, θ denotes a parameter of the Q-network, s' and a' are obtained from the experience replay pool.

Based on the obtained Critic-networks, the Actor-network π_ϕ is updated with a delay every few time-steps through maximizing the learned Q-function, where ϕ denotes the parameters of the Actor-network. Finally, TD3 delay updates the parameters of the three Target-networks:

$$\begin{aligned} \theta'_i &= \tau \theta_i + (1 - \tau) \theta'_i, \\ \phi' &= \tau \phi + (1 - \tau) \phi', \end{aligned} \quad (5)$$

where τ indicates soft update factor with a range of (0,1], θ'_i ($i = 1, 2$) denote the parameters of the two target Critic-networks, ϕ' is the parameter of the target Actor-network.

B. ASYNCHRONOUS RL

RL algorithms excel in exploring optimal policy. However, they employ a single-agent to update the parameters, which causes the agent to suffer from an overload of redundant information. This results in slow convergence of the model

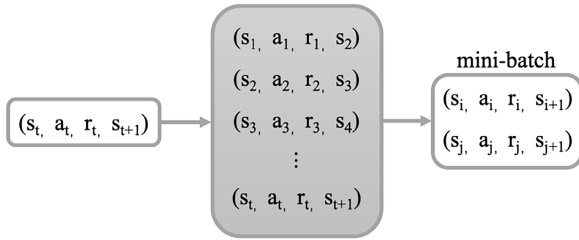


FIGURE 3. Experience Replay Mechanism. Train the network by randomly selecting a mini-batch of transitions from the experience replay pool.

and tends to fall into local optimums. To address these issues, scholars have incorporated the parallel update approach to the RL algorithms, which can be divided into two methods by using multiple agents that interact with the environment independently. The first method is synchronous parallelism, decomposing the state space into several subspaces and allowing each agent to explore among these subspaces. Another method is asynchronous parallelism, enabling each agent to explore the entire state space independently thus improving the convergence speed and performance through interaction information.

Fig. 1(b) and (c) illustrate the schematic of the two parallel models. Asynchronous parallelism has a time advantage over synchronous parallelism. While synchronous parallelism must wait for all agents complete the current iteration before proceeding to the next one, unlike asynchronous parallelism.

Asynchronous RL algorithms are derived from parallel RL, utilizing asynchronous parallelism to explore complex state spaces. Multiple agents learn independently and interaction information, thereby enhancing the exploration process [35]. When the state space is discrete, existing asynchronous RL algorithms suffer from limited convergence, as they fail to integrate properly with model-based approaches. In addition, when the state space is continuous, the asynchronous methods are integrated with Deep RL (DRL) algorithms, each agent transmits its gradient information to the global-network, which subsequently updates itself using the gradient information received from each agent.

Fig. 2 presents a classical asynchronous RL framework, known as Asynchronous Advantage Actor-Critic (A3C). The A3C algorithm consists of an Actor-network and a Critic-network. The Actor-network determines the action performed by the agent in next state, while the Critic-network evaluates the value of the action. Both networks operate independently and in parallel with each other. Specifically, the A3C algorithm trains multiple agents in parallel on single machine, thus significantly conserving hardware resources. During each iteration, individual agent interacts with its environment and transfers information to the global-network, which greatly accelerates the training process. However, the A3C algorithm is encumbered by several limitations. Firstly, it is susceptible to the complexity of the environment and the number of threads, which increases implementation and debugging challenges. Secondly, the algorithm demands

substantial data and computational resources, potentially constraining its adaptability and generalization in certain environments.

C. PRIORITIZED EXPERIENCE REPLAY MECHANISM BASED ON RANDOM SAMPLING

To address the temporal correlation of the experiences acquired by agents and the inability of agents to utilize historical experience, Lin [37] proposed the experience replay mechanism, as illustrated in Fig. 3.

Traditional RL algorithms based on experience replay [17], [18], [38] use uniform sampling. The probability of a sample being selected is described as follows:

$$p(i) = \frac{1}{n}, \quad (6)$$

where n denotes the number of samples in the experience replay pool.

However, uniform sampling prevents the agent from leveraging favorable samples for learning. To enhance learning efficiency, Van Seijen and Sutton [28] prioritized samples based on TD-errors, which enables samples with higher TD-error to have higher sampling probability, as follows:

$$\delta_t = R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t), \quad (7)$$

where t indicates the time-step, γ denotes a discount factor with a range of (0,1], Q denotes the state action-value function, S_t is the state at moment t , and A_t is the action executed at moment t .

Nevertheless, such TD error-based ranking methods suffer from the following problems:

- To avoid repeated scanning of the samples in the experience replay pool, traditional algorithms update the TD-error exclusively for the samples currently captured, which results in samples with initially low TD-errors being challenging to be sampled in subsequent training.
- The TD-error exhibits hysteresis, meaning that it is only associated with the current model. When the model parameters change, the TD-error adjusts accordingly.

Consequently, using Eq. (7) potentially overlooks samples that are informative for the current model.

To solve the above problem, [19] proposed a random sampling method that integrates pure greedy prioritization with uniform random sampling. This method ensures that the probability of a sample being selected is monotonic and non-zero in the prioritization process. Specifically, the probability of a sample i being selected as follows:

$$p(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad (8)$$

where $p_i = |\delta_t + \varepsilon|$ indicates the priority obtained by prioritizing sample i according to the TD-error, ε is a small value to ensure samples with a TD-error of zero are still selected with a non-zero probability, and α determines the weight of the experience replay, with $\alpha = 0$ indicating uniform sampling.

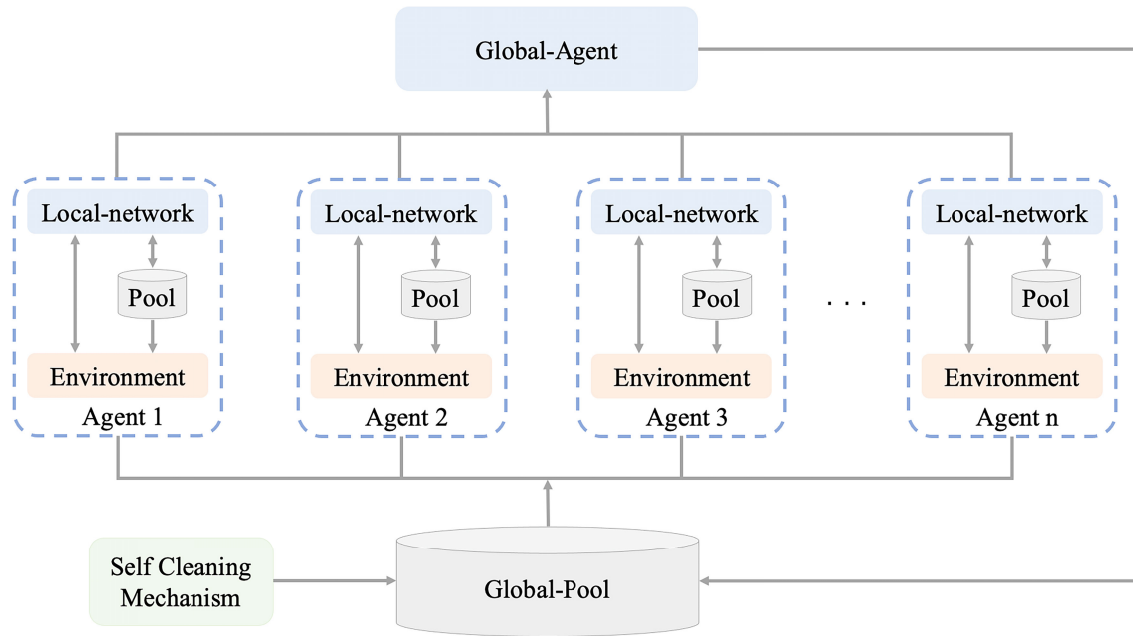


FIGURE 4. The network architecture of MP-TD3. The primary components comprise a global-agent, a global-pool, a self-cleaning mechanism and multiple agents.

III. MP-TD3 ALGORITHM

This section discusses the proposed MP-TD3 in detail. Initially, we analyze the challenges associated with prioritizing experience replay mechanism. Then, a multi-pool prioritized experience replay mechanism is proposed and further introduced into the TD3 algorithm. Subsequently, two global-pool self-cleaning mechanisms based on sample diversity and TD-errors, along with a self-updating mechanism and a multi-batch sampling mechanism are proposed.

A. PROBLEM FORMULATION

In an asynchronous RL framework, different agents interact independently with their respective environments. During each iteration, an agent initially observes the certain state s_t and selects an action a_t according to its policy. After executing the action, the environment delivers an immediate reward r_t to the agent and proceeds to the next state s_{t+1} . Meanwhile, the acquired samples $(s_t, a_t, s_{t+1}, r_t, done)$ are delivered into the respective experience replay pool, where *done* indicates the sign that the environment has reached the terminal state. During training, each agent selects a mini-batch of transitions from its respective experience replay pool and guides the model learning by calculating the TD-error, which updates the parameters of the global-agent until converges.

Traditional asynchronous RL algorithms employ uniform sampling, *i.e.*, the probability of selecting samples from each experience replay pool with $p = 1/n$. In this way, the importance of the information transmitted by various agents is neglected, leading agents to inadvertently prioritize communicating erroneous information to the global-network,

and influencing it to update parameters. A prioritized sampling structure intends to enable valuable samples to be sampled frequently by adjusting the sampling probability p , which improves the convergence speed and performance.

However, applying an experience replay mechanism directly to asynchronous RL raises the following issues:

1) During training, each agent selects samples only from its respective experience replay pool, which implies that an agent fails to utilize excellent experiences obtained by other agents interacting with the environment.

2) The asynchronous update mechanism leads to a significant similarity of the samples in the experience replay pool for each agent, which makes the samples suffer from an intense degree of redundancy.

3) The TD-error of the samples in the global-pool decreases with model updating, which results in a decrease in the information content of the samples.

4) Similar to experience replay in a single-agent, the problem of TD-error hysteresis exists in an asynchronous RL framework.

5) Integrating all the samples in experience replay pool corresponding to each agent generates a substantial volume of inter-process interactions.

To solve the above issues, we propose a Multi-pool Prioritized experience replay-based asynchronous Twin Delayed Deep Deterministic policy gradient algorithm (MP-TD3). Fig. 4 illustrates the network architecture of our MP-TD3, which consists of a global-agent that guides the update of each agent, a global-pool that aggregates samples from each experience replay pool and provides samples for each agent to train its network, a self-cleaning mechanism (SCM)

Algorithm 1 Multi-Pool Structure

Input: Initialize local-pools $p_{li}(i = 1, \dots, m)$, global-pool p_g ;
Initialize local-iteration n_l , global-iteration n_g ;
Initialize max iteration T , update frequency f ;
Initialize local-networks $M_{li}(i = 1, \dots, m)$ with random weight w' , global-network M_g with random weight w

Output: Global-network M_g

- 1: **while** $n_g \leq T$ **do**
- 2: Initialize state s_0
- 3: $cache \leftarrow \phi, t = 0$
- 4: **while** $done$ is False **do**
- 5: Select action with exploration noise $a_t \sim \pi_{\phi'}(s_t) + \epsilon, \epsilon \sim N(0, \sigma)$
- 6: Obtain reward r_t and next state s_{t+1}
- 7: Store transition tuple $(s_t, a_t, s_{t+1}, r_t, done, TD-error)$ in p_{li} and according to Eq. (8) in cache
- 8: $j \leftarrow n_l \% f$
- 9: **if** $n_g \leq l$ **then**
- 10: $b_j \leftarrow$ Sample mini-batch of transitions from p_{li}
- 11: **end if**
- 12: Compute the gradient $\nabla w'$ of M_{li} based on b_j
- 13: Asynchronous update the weights w of M_g based on the gradient $\nabla w'$
- 14: $n_g \leftarrow n_g + 1$
- 15: **if** $n_l \% f = 0$ **then**
- 16: Store samples in $p_g, cache \leftarrow \phi$
- 17: **end if**
- 18: **if** $n_g \geq l$ **then**
- 19: Sample f mini-batch $\{b_0, b_1, \dots, b_f\}$ from p_g
- 20: **end if**
- 21: $t+ = 1, n_g+ = 1, n_l+ = 1$
- 22: **end while**
- 23: **end while**
- 24: **return** Global-network M_g

that eliminates redundant samples from the global-pool, and multiple agents that offer gradient information to the global-agent.

B. MULTI-POOL PRIORITIZED EXPERIENCE REPLAY MECHANISM

To address the difficulty of each agent to utilize excellent experiences from other agents interacting with the environment, we propose a multi-pool structure, which is primarily comprised of multiple experience replay pools and a global-pool.

In RL algorithms, the samples with large TD-error tend to be more significant for model training. To leverage these samples, the proposed framework redefines the samples in the experience replay pool, *i.e.*, $(s_t, a_t, s_{t+1}, r_t, done, TD-error)$. At every time-step, each agent interacts independently with its respective environment and delivers excellent samples into its experience replay pool (*i.e.*, local-pool), which facilitates the agent to select such samples in the subsequent training, thus accelerating the convergence speed. Subsequently, the samples in the local pool are selected based on the probability of Eq. (8) and then transferred into the global-pool.

Furthermore, each agent requires numerous inter-process interactions, which increases the time overhead. To reduce the overhead of interactions, each agent samples directly from its local-pool in the early training stage. While in the later training stage, each agent samples from the global-pool rather than the respective experience replay pool. Compared with original TD3 algorithm, our MP-TD3 introduces a multi-pool structure. This design enables agents to select beneficial samples from other agents' experience replay pools, which improves the convergence speed and performance. Algorithm 1 summarizes the procedure of the proposed multi-pool structure.

C. SELF-CLEANING MECHANISM

To solve the intense degree of redundancy and low information content of samples, we propose two global-pool self-cleaning mechanism based on sample diversity and TD-errors, respectively.

1) SELF-CLEANING MECHANISM BASED ON SAMPLE DIVERSITY

Since each agent interacts with the environment independently and updates parameters through an asynchronous

Algorithm 2 Self-Cleaning Mechanism Based on Sample Diversity**Input:** global-pool p_g , delete number n , cluster number k **Output:** cleaned global-pool p_g

- 1: $T \leftarrow$ all samples in p_g
- 2: Concatenate *state* and *action* of each iteration form a point set P
- 3: Perform K-means algorithm to clustering P into k clusters: $\{c_i\}_{i=1}^k$
- 4: Record the size of each cluster $\{n_c^i\}_{i=1}^k$
- 5: Compute the average size of each cluster \bar{n}_c^i
- 6: Record the index of cluster in $\{c_i^*\}_{i=1}^k$ that are larger than the average size \bar{n}_c^i
- 7: $\mathbb{D} \leftarrow \phi$
- 8: **for** $i \leftarrow 1$ to α **do**
- 9: $d \leftarrow$ Compute the difference between c_i^* and \bar{n}_c^i
- 10: $\mathbb{D} \leftarrow \mathbb{D} \cup \{d\}$
- 11: **end for**
- 12: Allocate the delete number $\{n_i\}_{i=1}^\alpha$ of each cluster proportionally based on the difference d in \mathbb{D}
- 13: **for** $i \leftarrow 1$ to α **do**
- 14: Select n_i randomly from c_i^* and delete these samples from p_g
- 15: **end for**
- 16: **return** p_g

Algorithm 3 Self-Cleaning Mechanism Based on TD-Errors**Input:** global-pool p_g , delete number n **Output:** cleaned global-pool p_g

- 1: $T \leftarrow$ all samples in p_g
- 2: Order T in ascendant based on TD-errors and retain the index $T^* = \{T_i^*\}_{i=1}^n$ of the top n samples
- 3: **for** $i \leftarrow 1$ to n **do**
- 4: Remove T_i^* from p_g
- 5: **end for**
- 6: **return** p_g

updating mechanism, it may result in a high degree of sample similarity among the samples in the global-pool. It tends to select numerous similar samples when an agent selects samples from the global-pool, leading to a lack of diversity. To address this problem, we eliminate the samples with highly similar to reduce sample redundancy with the K-means algorithm. Specifically, we first concatenate the states s_t and actions a_t of the samples in the global-pool to form a new point set, then cluster point sets with the K-means algorithm to derive k categories, and finally, clean point sets in each cluster by a certain proportion. Algorithm 2 presents the procedure of the proposed self-cleaning mechanism based on sample diversity.

2) SELF-CLEANING MECHANISM BASED ON TD-ERRORS

During updating, the TD-error of samples tends to decrease progressively, resulting in a decrease in the low information content of samples and affecting the model performance. As a result, to ensure the high TD-error of the samples in the global-pool, we perform self-cleaning for the samples with minor TD-error. Algorithm 3 outlines the procedure for the self-cleaning mechanism based on TD-errors.

D. SELF-UPDATING MECHANISM BASED ON TD-ERRORS

The TD-error hysteresis of samples exists in asynchronous RL, in other words, the error information of the selected samples is significantly deviates from the actual situation, which influences the learning efficiency and performance. To tackle this issue, we propose two self-updating mechanisms for the global-pool and the experience replay pool (*i.e.*, local-pool) of each agent, respectively.

1) EXPERIENCE REPLAY POOL FOR EACH AGENT

During training, each agent selects a mini-batch of transitions from its respective experience replay pool for learning. Instead of iteratively scanning the experience replay pool, we update the TD-error only for the selected samples, which effectively reduces the time overhead and alleviates the TD-error hysteresis.

2) FOR GLOBAL-POOLS

To guarantee that each agent is able to select samples with high information content in the later training, the proposed method updates the TD-errors of the samples in the global-pool. Consequently, samples that initially exhibit high TD-errors but diminish after network updates can be eliminated by the self-cleaning mechanism. Ultimately, our method ensures that the samples beneficial for the current training are delivered in the global-pool with constrained capacity.

E. MULTI-BATCH SAMPLING MECHANISM

Since each agent selects samples from the global-pool, it increases the time overhead. To address this issue, we introduce a multi-batch sampling mechanism and a multi-cache structure.

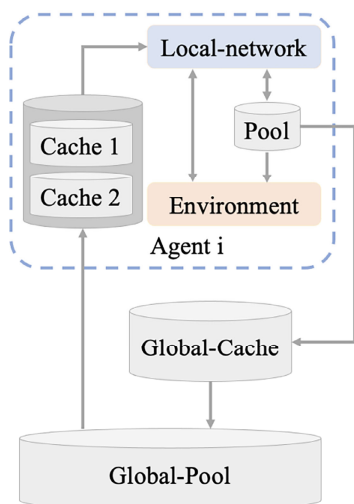


FIGURE 5. Multi-cache structure. It consists of a global-cache and two caches. The global-cache preserves samples transmitted from each experience replay pool, and each agent is sampled in respective cache.

1) MULTI-BATCH SAMPLING MECHANISM

The conventional sampling approach retrieves a mini-batch of transitions in a single operation. However, in the asynchronous approach, each agent needs to select samples from the global-pool, which consumes substantial process interaction time. Thus, we introduce a multi-batch sampling mechanism, that is, m_1 mini-batch of transitions are sampled from the global-pool at a time for agents to train m_1 iterations. In this way, the number of interactions between the agent and the global-pool can be reduced.

2) MULTI-CACHE STRUCTURE

Since Python is unable to be multi-processing and multi-threading such as C++, we design a multi-cache structure to reduce interaction time. As shown in Fig. 5, the designed multi-cache structure comprises a global-cache and two caches assigned to each agent. During the later stages of training, each agent needs to select samples from the global-pool. However, when multiple agents sample simultaneously, resource conflict arise.

Consequently, we allocate m_2 caches to each agent so that each agent only needs to sample from its individual cache. To enable the agent to adopt as many new samples as possible as well as to avoid resource conflict, m_2 is assumed to 2. Similarly, the global-pool requires to sample from the experience replay pool of each agent, which leads to the identical issue. As a result, a global-cache is designed to store the samples transmitted from each experience replay pool. Once there are available samples in the global-cache, they are delivered into the global-pool.

IV. EXPERIMENT AND ANALYSIS

We compared the performance of the proposed MP-TD3 with TD3 [36], DDPG [18], Proximal Policy Optimization (PPO) [39], Vanilla Policy Gradient (VPG) [40], A3C [15] and Soft

TABLE 1. Settings and hyperparameters.

Symbol	Hyperparameters	Value
-	capacity	1000000
ϵ	exploration-noise	0.25
-	mini-batch size	100
γ	discount factor	0.99
τ	soft update factor	0.005
α	learning rate	0.0001
β	beta	0.99
U	delay-update	2
m_1	delay-sample	1000
m_2	number of cache	2

Actor-Critic (SAC) [41]. The details of TD3 and A3C are discussed in Sub-section II-A and II-B respectively. DDPG employs value functions approximators and introduces a model-free actor-critic algorithm that utilizes deep function approximation to learn policies in high-dimensional and continuous action spaces. PPO samples data through interactions with the environment and adopts stochastic gradient ascent to optimize the objective function. VPG approximates a value function to determine a policy, which is represented by its own function approximator with independent of the value function, and is updated according to the gradient of expected reward with respect to the policy parameters. SAC employs an actor-critic maximum entropy RL framework, the actor intends to maximize the expected reward while maximizing entropy.

A. EXPERIMENTAL ENVIRONMENT

To evaluate the proposed MP-TD3, we measured its performance through a set of MuJoCo continuous control tasks [42], which are interfaced via OpenAI Gym [43]. In addition, the original task set designed by Brockman et al. [43], without modifications, is used to facilitate comparison with other algorithms.

B. EXPERIMENTAL SETTING

In the proposed MP-TD3, both actor and critic networks are designed with two hidden-layers comprising exponential linear units (ELUs), each containing 256 neurons. Note that the actor-network employs a tanh activation unit before outputting the result. Furthermore, the Adam [44] optimizer is used to update the parameters of both networks.

At each time-step, the network is trained by uniformly sampling the mini-batch of transitions from the experience replay pool. For a fair comparison, the proposed MP-TD3 utilizes the identical hyperparameters as the original TD3. Table 1 presents the hyperparameter settings used throughout our experiments. A series of experiments are performed to validate the effectiveness of the proposed MP-TD3.

The experiments are conducted on a computer equipped with Intel(R) Xeon(R) gold 6152 CPU@2.10GHz, Tesla

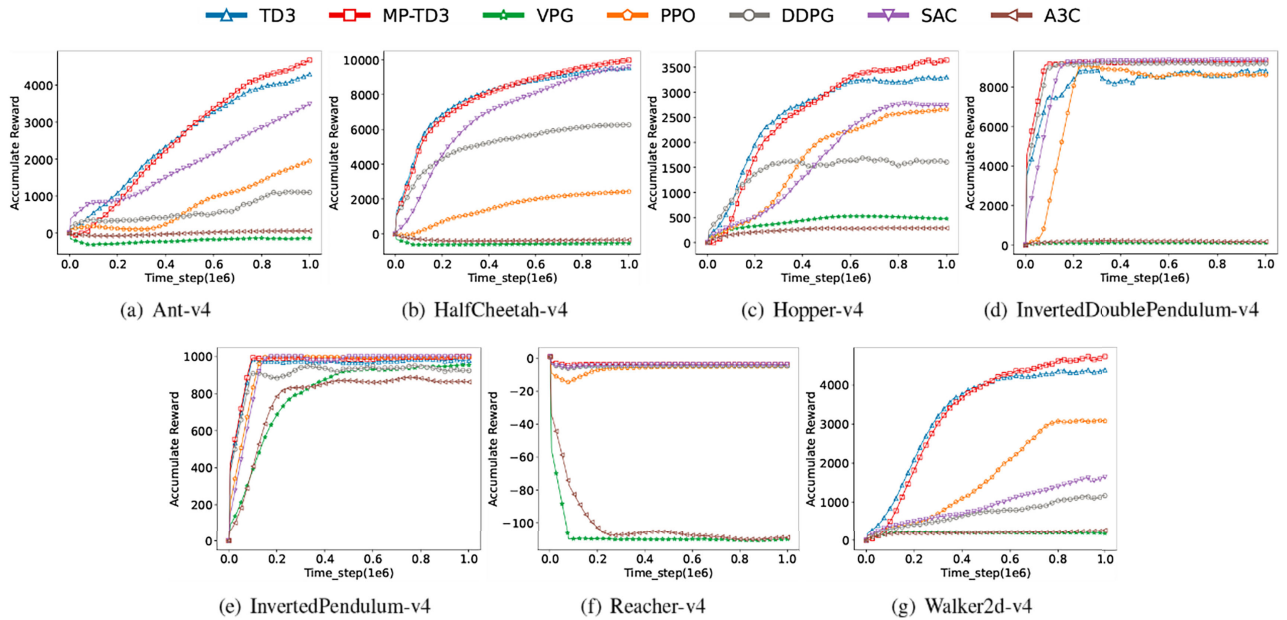


FIGURE 6. The results of different algorithms in each environment for 1 million time-steps.

V100 32GB and a maximum of 256GB of RAM. The algorithms are implemented by Python, compiled by PyCharm under a Linux operating system.

C. COMPARISON EXPERIMENT

Each algorithm is executed for 1 million time-steps with evaluations every 5000 time-steps. In addition, exploration noise is removed during testing, and each algorithm is evaluated in at least 10 trials to ensure accurate results.

Fig. 6 illustrates learning curves of MP-TD3 and other state-of-the-art algorithms over 1 million time-steps. The off-policy algorithms, *i.e.*, MP-TD3, TD3, SAC, and DDPG, are ranked as the top three 20 times out of 21. In contrast, the on-policy algorithms, *i.e.*, PPO, VPG, and A3C, are ranked as the top three only 1 time out of 21. Moreover, in the first 500,000 time-steps, the off-policy algorithms are ranked as the top three 19 times out of 21, while the on-policy algorithms are ranked as the top three only 2 times out of 21. This indicates that off-policy algorithms are superior in convergence speed and performance. Such algorithms leverage the experience replay pool to store the experiences accumulated by agents interacting with the environment, so that the agents are able to update the model based on a blend of previous and recent experiences. On the one hand, this mechanism mitigates the correlation between training data. On the other hand, it facilitates the repeated sampling of high-value experiences, which improves the learning efficiency and stability.

In addition, we compare the performance of seven algorithms, including three on-policy and four off-policy algorithms. The results reveal a marked superiority of the off-policy algorithms. Specifically, the off-policy algorithms ranked top 5/10/20 times across all the environments, while

the on-policy algorithms ranked top 0/10/0 times. During the initial 500,000 time-steps, the off-policy algorithms ranked top 5/10/20 times, while the on-policy algorithms remained at 0/10/0. Consequently, the off-policy algorithms demonstrate significant advantages in RL.

Among the four off-policy algorithms, our MP-TD3 achieves the top ranking on five occasions. In comparison, SAC, TD3 and DDPG ranked top 2/10/0 respectively. Overall, the final results of MP-TD3 and TD3 are significantly better than those of SAC and DDPG. This superiority can be attributed to the dual-q clip mechanism and the delayed update mechanism, which effectively mitigates over-estimation and policy degradation. However, our MP-TD3 performs better than the original TD3. This is attributed to introducing asynchronous updating and the proposed multi-pool prioritized experience replay mechanism, which stimulates the experience interactions among different agents, thus improving the convergence speed and performance.

It is noteworthy that in the Reacher environment (see Fig. 6(f)), the final cumulative rewards obtained by other algorithms exhibit negligible differences, with the exception for the VPG and A3C algorithms. This likely stems from the relative simplicity of the Reacher environment, where most algorithms are able to explore the optimal policy rapidly.

To further evaluate the performance of the proposed MP-TD3, Table 2 depicts the cumulative rewards obtained by each algorithm across seven environments from 10^5 to 10^6 time-steps, where red, blue and green indicate the first, second and third ranked algorithms respectively. As can be seen from Table 2, our MP-TD3 exhibits the suboptimal performance compared with other algorithms in the early stages. This can be attributed to the implementation of an asynchronous updating mechanism, each agent updates

TABLE 2. Cumulative rewards obtained after 1e5-1e6 time-steps from learning different methods.

Environment	Algorithm	Time-step									
		100000	200000	300000	400000	500000	600000	700000	800000	900000	1000000
Ant	DDPG	358.72	154.84	350.46	499.42	568.94	415.52	628.46	976.35	1354.78	1047.49
	VPG	-285.38	-237.21	-199.21	-366.1	-197.33	-179.03	-174	-110.4	-226.45	-98.99
	PPO	151.76	123.05	74.15	148.93	621.55	1057.91	1114.74	1413.95	1734.28	1939.92
	A3C	-76.22	-83.79	-58.11	-37.45	-7.45	14.72	32.54	44.81	49.95	53.15
	SAC	802.09	851.67	1027.79	1532.87	1838.24	1880.37	2547.06	2495.49	3334.39	3368.29
	TD3	585.51	931.22	1953.46	1889.38	2890.34	3495.15	3661.2	4055.17	3677.18	4242.53
MP-TD3	207.27	807.03	1578.27	2229.05	2845.43	3375.69	3847.38	4207.12	4385.42	4677.57	
HalfCheetah	DDPG	3323.67	4492.06	4929.78	5083.41	5651.93	5850.15	5944.6	6059.56	6205	6356.5
	VPG	-632.85	-623.83	-616.75	-592.63	-587.61	-598.08	-587.12	-562.4	-566.34	-544.51
	PPO	52.2	810.26	1154.62	1446.51	1754.09	2012.43	2180.52	2270.47	2364.16	2428.57
	A3C	-316.17	-422	-423.01	-415.61	-413.19	-397.96	-388.49	-381.15	-361.82	-354.03
	SAC	1916.9	4507.12	6297.03	7061.4	7624.97	8014.92	8659.66	9007.41	9412.23	9700.36
	TD3	5559.13	6918.57	7820.07	8207.95	8538.62	8804.61	9039.99	9129.95	9539.56	9635.66
MP-TD3	4738.02	6580.82	7497.07	8108.32	8572.2	8928.64	9251.98	9559.36	9791.47	9979.84	
Hopper	DDPG	818.58	1914.69	1630.34	1439.78	1404.19	1572.64	1851.39	1481.15	1338.56	1455.46
	VPG	271.79	327.94	359.96	440.49	532.32	507.51	529.58	512.01	470.78	491.78
	PPO	293.4	484.9	915.96	1824.44	2142.46	2236.37	2499.99	2654.86	2653.72	2696.08
	A3C	167.13	209.86	235.36	266.24	290.3	275.52	285.71	286.64	284.44	287.59
	SAC	311.27	579.01	860.79	1332.64	2123.15	2318.7	2471.15	2884.48	2729.91	2923.88
	TD3	659.45	2441.1	2410.35	2765.82	2933.26	3047.91	3262.75	3331.74	3368.61	3390.79
MP-TD3	470.91	1668.38	2324.32	2670	2959.9	3305.32	3442.46	3463.52	3608.83	3641.49	
Inv-D-Pen	DDPG	9219.52	9298.34	9270.37	9244.12	9234.48	9226.21	9223.18	9214.97	9202.16	9204.55
	VPG	86.75	83.96	84.5	88.92	97.9	83.16	98.89	98.52	94.39	92.16
	PPO	542.71	8982.57	8751.09	9106.4	9004.53	7970.36	8672.18	8327.62	9128.07	8660.65
	A3C	146.91	185.45	200.33	187.6	179.97	168.87	189.87	141.57	130.56	156.63
	SAC	8798.97	9272.07	9275.89	8838.19	9353.04	9354.82	9355.7	9356.14	9356.79	9358.51
	TD3	6923.43	9022.36	8519.31	7495.69	8460.78	7766.53	9324.21	8559.83	9256.05	8562.36
MP-TD3	9149.38	9242.7	9287.87	9230.48	9257.36	9264.63	9280.66	9259.29	9280.8	9292.63	
Inv-Pen	DDPG	1000	793.25	923.1	1000	809.03	1000	1000	1000	1000	912.61
	VPG	414.39	742.84	847.75	904.56	910.79	919.48	909.44	911.05	943.75	951.9
	PPO	1000	1000	1000	1000	1000	960.55	990.09	990.21	1000	1000
	A3C	419.62	821.31	827.59	865.16	865.99	862.52	880.34	886.68	857.91	868.46
	SAC	1000	1000	1000	903.86	1000	1000	1000	1000	1000	1000
	TD3	1000	1000	1000	1000	912.15	1000	1000	1000	928.72	1000
MP-TD3	1000	1000	1000	1000	1000	986.77	1000	1000	995.79	1000	
Reacher	DDPG	-5.85	-4.23	-4.01	-4.53	-4.82	-4	-4.12	-3.96	-4.27	-4.09
	VPG	-111.88	-108.92	-110.17	-109.42	-112.28	-110.73	-111.75	-111.01	-110.99	-110.82
	PPO	-13.77	-6.43	-6	-5.12	-4.72	-5.57	-4.78	-4.78	-4.78	-4.84
	A3C	-83.65	-107.52	-108.25	-105.95	-105.3	-109.21	-107.64	-110.58	-110.46	-108.42
	SAC	-4.69	-4.24	-3.82	-3.76	-3.91	-3.95	-3.61	-3.91	-3.58	-3.7
	TD3	-4.63	-3.88	-4.08	-4.25	-4.11	-4.15	-4.24	-4.23	-3.95	-4.22
MP-TD3	-4.04	-3.7	-3.66	-3.66	-3.61	-3.67	-3.68	-3.58	-3.67	-3.68	
Walker2d	DDPG	323.87	513.49	581.21	705.46	634.26	879.64	932.35	875.3	1209.09	1208.97
	VPG	204.03	206.15	213.07	209.34	206.51	199.22	177.05	204.19	182.86	202.94
	PPO	348.06	432.61	654.83	1033.86	1471.02	2336.13	2744.66	3236.41	2813.75	3064.13
	A3C	201.75	177.45	187.33	205.1	205.94	214.89	215.02	217.24	247.63	259.84
	SAC	376.66	480.84	722.6	704.73	660.12	1069.04	1362.07	1365.25	1479.19	1841.67
	TD3	770.15	2420.88	3335.47	3735.66	4195.8	4362.68	4441.72	4420.48	3673.98	4412.89
MP-TD3	506.77	1817.12	3013.83	3666	4043.13	4298.31	4424.69	4618.53	4676.58	4725.03	

Note that, for each time-step, red, blue and green represent the first, second and third ranked algorithms, respectively.

parameters at varying rates, thus leading to instability during the initial stage. However, upon reaching a certain time-steps, our MP-TD3 attains greater stability. In addition, the performance metrics across the seven environments over ten iterations are documented, encompassing a total of 70 comparisons. Our MP-TD3 is ranked among the top three algorithms for 69 times and ranked as the first algorithm for 37 times out of 70 regarding time-step, respectively. This indicates that our MP-TD3 exhibits great performance

and competitiveness compared with other state-of-the-art methods in different environments.

To evaluate the computational complexity of our MP-TD3, we compare Floating Point Operations (FLOPs) required to train a time-step for the proposed MP-TD3 and the original TD3 in seven environments. As shown in Table 3, the proposed MP-TD3 suffers from higher FLOPs than the original TD3. This is due to the introduction of the multi-pool structure in our MP-TD3, in which each agent selects samples

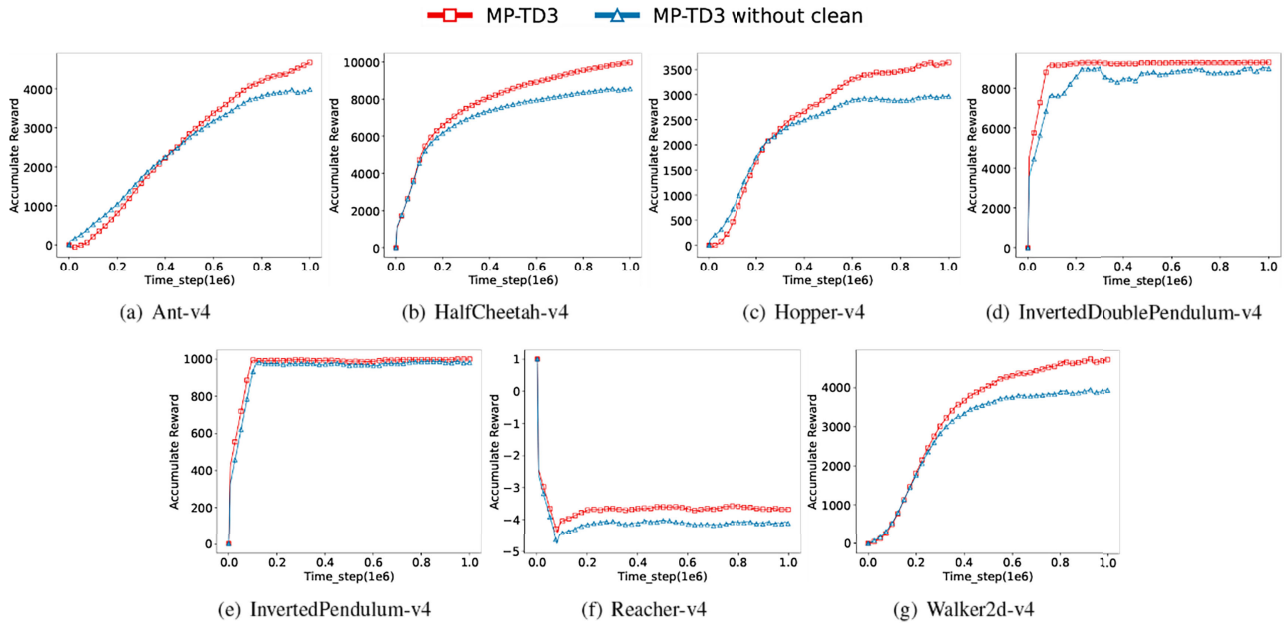


FIGURE 7. The effectiveness of self-cleaning mechanisms compared.

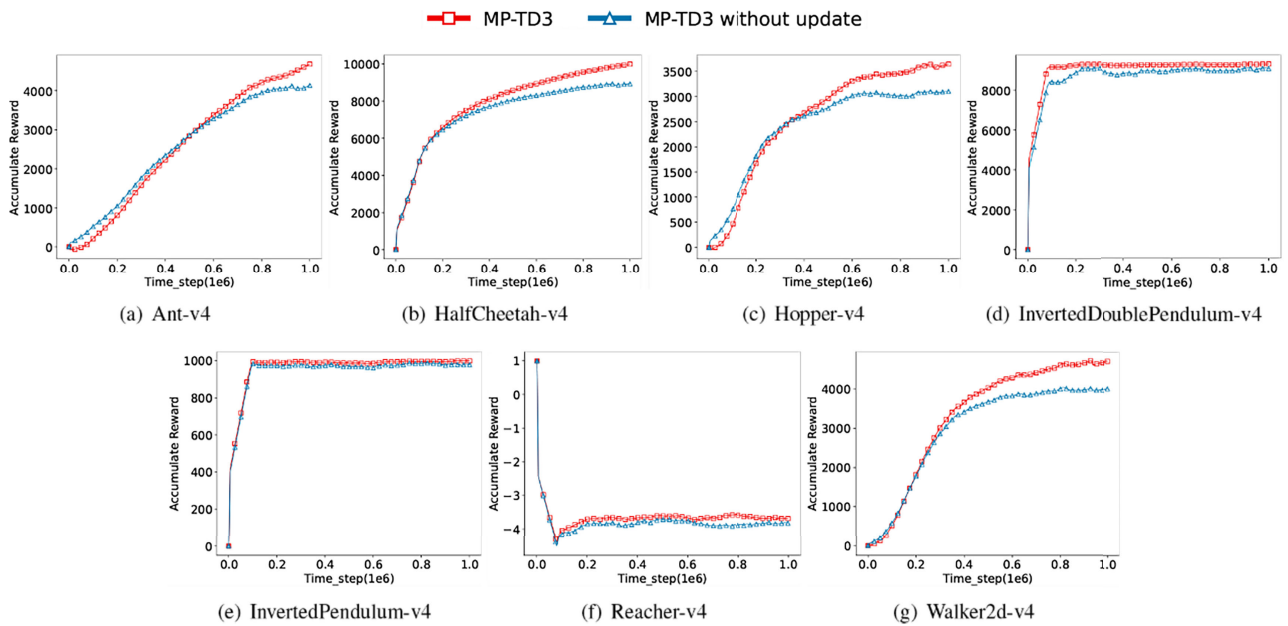


FIGURE 8. The effectiveness of self-updating mechanisms compared.

not only from its respective experience replay pool, but also directly from the global-pool. Moreover, to prioritize the samples in each time-step, it is necessary to compute their TD-errors. However, our MP-TD3 is superior to the original TD3 in convergence speed and performance. Consequently, the proposed method is still considerably competitive.

In summary, the above experimental results indicate that:

- The proposed our MP-TD3 possesses fast convergence speed. It is due to the incorporation of a multi-pool prioritized experience replay mechanism,

which enables each agent to utilize excellent experiences acquired by other agents interacting with the environment, thus facilitating the acquisition of high-value samples.

- The proposed our MP-TD3 achieves superior cumulative rewards in several benchmark environments. It is attributed to the implementation of two self-cleaning and self-updating mechanisms, which maintain the diversity and high information content of the samples delivered in the experience replay pool.

TABLE 3. The FLOPs for the original TD3 and the proposed MP-TD3.

Environment	TD3	MP-TD3 (ours)
Ant	144.307 M	182.809 M
HalfCheetah	107.443 M	136.115 M
Hopper	103.982 M	131.737 M
Inv-D-Pen	103.219 M	130.764 M
Inv-Pen	100.531 M	127.360 M
Reacher	103.603 M	131.251 M
Walker2d	107.443 M	136.115 M

D. ABLATION EXPERIMENT

We further conducted ablation experiments to explore the effectiveness of the proposed MP-TD3.

1) THE EFFECTIVENESS OF SELF-CLEANING MECHANISM

- MP-TD3 without clean: MP-TD3 without self-cleaning mechanism;
- MP-TD3: the final MP-TD3.

To evaluate the effectiveness of the proposed self-cleaning mechanism, we compare the performance of MP-TD3 and MP-TD3 without self-cleaning mechanism. From Fig. 7, it is evident that MP-TD3 is significantly more effective than MP-TD3 without clean in most environments, with MP-TD3 featuring higher cumulative rewards. This is attributed to the use of the proposed self-cleaning mechanism, which reduces the similarity among the samples. Consequently, agents are able to acquire more diverse samples, contributing to the ability of the model to learn better in complex environment and improving performance. Furthermore, the self-cleaning mechanism enhances the information content of the samples through removing redundant samples from the global-pool, which means that agents acquire more beneficial samples and accelerates the convergence speed. In general, compared with the traditional TD3 algorithm, the proposed MP-TD3 is able to leverage excellent experience, which accelerates the convergence speed.

2) THE EFFECTIVENESS OF SELF-UPDATE MECHANISM

- MP-TD3 without update: MP-TD3 without the self-update mechanism;
- MP-TD3: the final MP-TD3.

To evaluate the effectiveness of the proposed self-updating mechanism, we compare the performance of the proposed MP-TD3 and the MP-TD3 without self-updating mechanism. As illustrated in Fig. 8, the performance of MP-TD3 is significantly improved compared to MP-TD3 without update in most environments, with MP-TD3 featuring higher cumulative rewards. This is due to the utilization of self-updating mechanism, which enables the samples in the global-pool to maintain the most recent TD-error constantly. Therefore, the agents are able to select samples with the maximum

global TD-error for training, effectively accelerating the convergence speed. Moreover, the self-updating mechanism enables the agents to prioritize samples with high information content. As the samples frequently update TD-error in the global-pool, it is ensured the samples that are more beneficial for the current training are stored as much as possible in the limited capacity of global-pool, thus improving the efficiency of the model. In a word, compared with the traditional TD3, MP-TD3 utilizes a self-updating mechanism that assures the real-time and usefulness of the samples, which significantly improves the convergence speed and performance.

V. CONCLUSION

To tackle the problem of slow model convergence caused by applying the prioritized experience replay mechanism to asynchronous RL, we propose a Multi-pool Prioritized experience replay-based asynchronous Twin Delayed Deep Deterministic policy gradient algorithm (MP-TD3). To stimulate the experience interactions among different agents, a multi-pool prioritized experience replay mechanism is designed. Subsequently, a self-cleaning mechanism based on sample diversity and a self-cleaning mechanism based on TD-errors are separately designed to remove redundant samples and preserve the diversity of samples. Furthermore, a self-updating mechanism based on TD-errors is proposed to alleviate the TD-error hysteresis. Experimental results demonstrate that the proposed MP-TD3 significantly accelerates the convergence of asynchronous RL with competitive performance compared to state-of-the-art methods. However, our MP-TD3 suffers from high computational complexity. Consequently, in our future work we will focus on the following directions: a) develop a more efficient asynchronous method to further improve convergence efficiency; b) simplify the MP-TD3 framework to reduce computational costs.

REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
- [2] L. Lyu, Y. Shen, and S. Zhang, "The advance of reinforcement learning and deep reinforcement learning," in *Proc. IEEE Int. Conf. Electr. Eng., Big Data Algorithms (EEBDA)*, Feb. 2022, pp. 644–648.
- [3] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annu. Rev. Control, Robot., Auto. Syst.*, vol. 5, no. 1, pp. 411–444, May 2022.
- [4] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: Lessons we have learned," *Int. J. Robot. Res.*, vol. 40, no. 4–5, pp. 698–721, Apr. 2021.
- [5] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, Apr. 2023.
- [6] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.
- [7] J. Wu, Z. Huang, and C. Lv, "Uncertainty-aware model-based reinforcement learning: Methodology and application in autonomous driving," *IEEE Trans. Intell. Vehicles*, vol. 8, no. 1, pp. 194–203, Jan. 2023.
- [8] R. N. Boute, J. Gijsbrechts, W. van Jaarsveld, and N. Vanvuchelen, "Deep reinforcement learning for inventory control: A roadmap," *Eur. J. Oper. Res.*, vol. 298, no. 2, pp. 401–412, Apr. 2022.

- [9] J. Gijsbrechts, R. N. Boute, J. A. Van Mieghem, and D. J. Zhang, "Can deep reinforcement learning improve inventory management? Performance on lost sales, dual-sourcing, and multi-echelon problems," *Manuf. Service Operations Manage.*, vol. 24, no. 3, pp. 1349–1368, May 2022.
- [10] F. Rupp, M. Eberhardinger, and K. Eckert, "Balancing of competitive two-player game levels with reinforcement learning," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2023, pp. 1–8.
- [11] K. Souchleris, G. K. Sidiropoulos, and G. A. Papakostas, "Reinforcement learning in game industry—Review, prospects and challenges," *Appl. Sci.*, vol. 13, no. 4, p. 2443, Feb. 2023.
- [12] C. J. C. H. Watkins, "Learning from delayed rewards," *Robot. Auton. Syst.*, vol. 15, no. 4, pp. 233–235, 1989.
- [13] O. Sigaud and O. Buffet, *Markov Decision Processes in Artificial Intelligence*. Hoboken, NJ, USA: Wiley, 2013.
- [14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.
- [15] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1008–1014.
- [16] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 293–321, May 1992.
- [17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
- [18] 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," 2015, *arXiv:1509.02971*.
- [19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.
- [20] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1243–1253, Feb. 2019.
- [21] G. Dao and M. Lee, "Relevant experiences in replay buffer," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2019, pp. 94–101.
- [22] J. Yang and G. Peng, "DDPG with meta-learning-based experience replay separation for robot trajectory planning," in *Proc. Int. Conf. Control, Autom. Robot. (ICCAR)*, Apr. 2021, pp. 46–51.
- [23] Q. Wei, H. Ma, C. Chen, and D. Dong, "Deep reinforcement learning with quantum-inspired experience replay," *IEEE Trans. Cybern.*, vol. 52, no. 9, pp. 9326–9338, Sep. 2021.
- [24] S. Sinha, J. Song, A. Garg, and S. Ermon, "Experience replay with likelihood-free importance weights," in *Proc. Learn. Dyn. Control Conf.*, 2022, pp. 110–123.
- [25] X. Liu, T. Zhu, C. Jiang, D. Ye, and F. Zhao, "Prioritized experience replay based on multi-armed bandit," *Expert Syst. Appl.*, vol. 189, Mar. 2022, Art. no. 116023.
- [26] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Mach. Learn.*, vol. 13, no. 1, pp. 103–130, Oct. 1993.
- [27] D. Andre, N. Friedman, and R. Parr, "Generalized prioritized sweeping," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 10, 1997, pp. 1001–1007.
- [28] H. Van Seijen and R. Sutton, "Planning by prioritized sweeping with small backups," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 361–369.
- [29] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.
- [30] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.
- [31] S. Fujimoto, D. Meger, and D. Precup, "An equivalence between loss functions and non-uniform sampling in experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 14219–14230.
- [32] M. Brittain, J. Bertram, X. Yang, and P. Wei, "Prioritized sequence experience replay," 2019, *arXiv:1905.12726*.
- [33] X. Cao, H. Wan, Y. Lin, and S. Han, "High-value prioritized experience replay for off-policy reinforcement learning," in *Proc. IEEE 31st Int. Conf. Tools with Artif. Intell. (ICTAI)*, Nov. 2019, pp. 1510–1514.
- [34] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
- [35] X. Zhao, S. Ding, and Y. An, "A new asynchronous architecture for tabular reinforcement learning algorithms," in *Proc. Int. Conf. Extreme Learn. Mach. (ELM)*, Oct. 2019, pp. 172–180.
- [36] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
- [37] L.-J. Lin, "Programming robots using reinforcement learning and teaching," in *Proc. 9th Nat. Conf. Artif. Intell.*, 1991, pp. 781–786.
- [38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
- [40] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1057–1063.
- [41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
- [42] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.
- [43] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.
- [44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.



WENWEN TAN is currently pursuing the B.Eng. degree with the College of Engineering, Huaqiao University, Quanzhou, China. Her research interests include reinforcement learning and its related applications.



DETIAN HUANG (Member, IEEE) received the B.S. degree in electronic information engineering from Xiamen University, China, and the Ph.D. degree in circuits and systems from the University of Chinese Academy of Sciences, China. He is currently an Associate Professor with the College of Engineering, Huaqiao University. He has published more than 40 papers in well-known journals and conferences, including one Best Student Paper Award in the International Symposium on Intelligent Signal Processing and Communication Systems 2022 (ISPACS 2022). His research interests include computer vision, image restoration, object tracking, and deep learning.

...