ITC 4/54
Information Technology and Control
Vol. 54 / No. 4/ 2025
pp. 1159-1177
DOI 10.5755/j01.itc.54.4.40496

Efficient Training Framework for Multi-USV System Based on Off-policy Deep Reinforcement Learning

Received 2025/02/14 | Accepted after revision 2025/07/13

HOW TO CITE: Chen, Z., Li, H., Yan, B. (2025). Efficient Training Framework for Multi-USV System Based on Off-policy Deep Reinforcement Learning. *Information Technology and Control*, 54(4), 1159-1177. https://doi.org/10.5755/j01.itc.54.4.40496

Efficient Training Framework for Multi-USV System Based on Off-policy Deep Reinforcement Learning

Zhuoying Chen, Huiping Li*
School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an, 710072, China;
e-mail: czysmile@mail.nwpu.edu.cn, lihuiping@nwpu.edu.cn

Bing Yan
School of Electrical and Mechanical Engineering, The University of Adelaide, Adelaide, SA, Australia;
e-mail: bing.yan@adelaide.edu.au

Corresponding author: lihuiping@nwpu.edu.cn

Prioritized Experience Replay (PER) is a technical means of off-policy deep reinforcement learning by selecting important experience samples more frequently to improve the training efficiency. However, the non-uniform sampling applied in PER inevitably shifts the state-action distribution and brings the estimation errors of Q-value function. In this paper, an efficient off-policy reinforcement learning training framework called Attention Loss Adjusted Prioritized (ALAP) Experience Replay, is proposed. ALAP exploits the similarity of the transitions in buffer to quantify the training progress, and accurately corrects the bias based on the positive correlation between the error compensation strength and the training progress. In order to verify the effectiveness of the algorithm, the ALAP is tested on 15 different games on Atari 2600 benchmark. Additionally, we developed a multi-USV competition scenario using Unreal Engine to further illustrate the superiority as well as the practical value of ALAP.

KEYWORDS: Prioritized Experience Replay, Off-Policy Deep Reinforcement Learning, Attention, Multi-USV

1. Introduction

In recent years, deep reinforcement learning (DRL) [29] has achieved significant achievements in various domains, including go [25], multi-agent combat [2], robot control [8-9], etc. Experience Replay [13] is a critical training technique used in off-policy DRL. It learns the neural network by randomly selecting a fixed number of experience samples from buffer, and breaks the relevance between them. Experience Replay stores the samples in the experience pool with the same sampled frequency. This uniform sampling approach makes the agent perform more trial-and-error to find out samples with learning value, which cause too much invalid exploration in the early stage of training. Moreover, when the agent is in the sparse reward environment [1], this disadvantage is rather more obvious, making the algorithm difficult to converge.

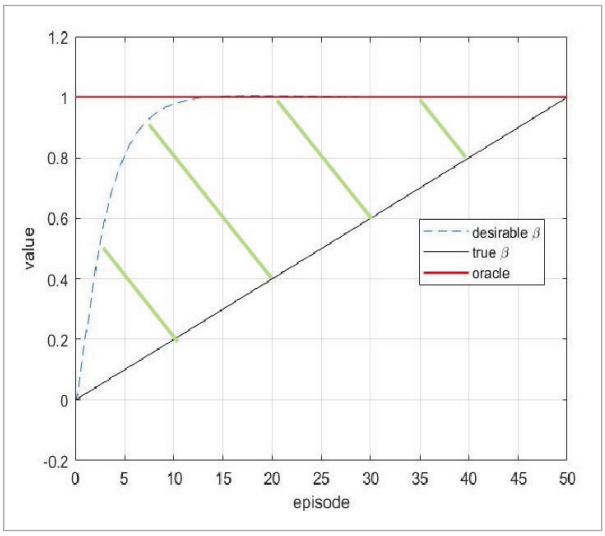
To overcome the limitations of the aforementioned method, the Priority Experience Replay (PER) mechanism is proposed [21]. Its core idea is to take the absolute value of the Temporal Difference (TD) error as the index to measure the importance of each sample. By artificially changing the sampled probabilities of the collected transitions in buffer and learning the relatively important one more frequently, the training efficiency of the network has been improved. However, it is noted that the non-uniform sampling of PER shifts the distribution that is originally used to estimate the Q-value function [18, 17, 23, 28], which leads to additional estimation error [32].

Aiming at error correction, a hyperparameter β [21, 26] is introduced to control the error compensation strength by regulating the weight of importance sampling, which can prevent the instability during training. Note that β. is positively correlated with training progress [21], and it increases linearly to 1 from initial value β_0 as the number of training episodes increases. However, the training progress is not distributed uniformly over training episodes, and a linear change of β would introduce extra errors. We offer the further illustration about this situation in Figure 1.

The initial value of β is assumed to be 0. Ideally, the trend of β should be consistent with the progress of convergence, which is shown as the desirable β. The oracle represents our training goal with highest

Figure 1

The incorrect setting of β.



reward 1. It is clear that the system is close to convergence after 10 episodes of training. However, the actual β now is linearly set to be 0.2, much less than the desired value of 1 for the converged model, which would cause large estimation errors of Q-value function during training.

To resolve this issue, in this paper, an efficient and general training framework called Attention Loss Adjusted Prioritized (ALAP) Experience Replay is proposed. ALAP can greatly reduce the estimation error of the Q-value function from both active and passive aspects. On one hand, the dynamic loss function is formulated based on Huber equation [11, 4, 20], which passively adapts to varying magnitudes of TD errors. On the other hand, an innovative self-attention variant called Shuffling Self-Attention network (S2AN) is designed to fit β precisely.

The attention module can actively measure the similarity of sample distribution from experience pool, which is used to quantify the exact training process. To establish the mapping relationship between β and the output of attention module, we normalize the output to achieve a more accurate β. This approach actively reduces the estimation error associated with the Q-valued function.

Furthermore, the priority based sampling (PS) of PER applies the fixed criterion for experience selection, which cannot restore the original sample distri-

bution of experience pool that required by attention module. As a result, we propose a Double-Sampling mechanism that covers both PS and random uniform sampling (RUS) simultaneously. PS is only in charge of providing learning samples for model training, while RUS is responsible for offering data input of attention module.

To evaluate the efficiency of our training framework, we first compare ALAP with the conventional PER in Atari 2600 benchmark, and ALAP outperforms much better than PER in most scenarios. To further demonstrate its practical value, we construct a multi-USV pursuit-evasion game, and make Multi-agent Twin Delayed Deep Deterministic Policy Gradient (MATD3) [31] as the algorithm carrier to test each training method. During training, ALAP outperforms existing PER variants, and exhibits the fastest convergence rate.

2. Related Work

The inspiration of priority sampling used in reinforcement learning comes from the prioritized sweeping for value function iterations, and its success in DQN [19, 3, 22] has attracted a lot of attention. DQN uses the same maximization operator to select and evaluate actions, which causes the problem of overestimation. Subsequently, PER began to be applied on Double-DQN (DDQN) [27]. Additionally, PER has generated favorable outcomes in many algorithms such as DDPG [7, 16], and Rainbow [6], etc.

PER itself has also evolved many improved versions. Liu and Zou [14] and Vanseijen and Sutton [30] studied the effect of buffer size on algorithm performance. In order to alleviate the waste of computing resources caused by the excessive experience pool, Shen et al. [24] proposed an experience classification method, which divides the TD error distribution into different segments, and the transition tuples in the same segment have the same priority. This clustering method can reduce the cache space, and break the relevance of experience samples. Aiming to screen out more valuable experience samples, Gao et al. [5] integrated the reward value with TD error to form a new priority parameter, which replaces TD error as the screening basis.

The aforementioned works mainly focus on the adjustment of sample priority evaluation, while ignoring the deviation caused by PER itself. In order to resolve this problem, a Loss Adjusted Prioritized (LAP) [4] algorithm is proposed, which proves that any loss function evaluated by non-uniform sampling can be transformed into another uniform sampling one with the same expected gradient. The loss function is segmentally described by Huber function, and different sampling methods are adopted according to the variation of TD error. This can suppress the sensitivity of the root Mean Square Error (MSE) loss to outliers, and reduce the influence of Mean Absolute Error (MAE) loss on convergence rate. On the basis of LAP, the work [20] theoretically analyzes the reasons for the poor effect on the combination of PER and actor-critic algorithm in continuous action control environment. It claims that there is a significant error between Q-function calculation and the actual value, so it cannot correctly guide the actor network to make reasonable actions.

The LAP algorithm improves the robustness against outliers by designing the segmented loss function. However, the non-uniform sampling part still changes the state-action distribution for Q function estimation when the MAE loss is applied, resulting in the estimation deviation. Therefore, LAP does not solve the underlying problem of PER.

In this paper, we propose a new method to resolve this issue by accurately determining the relation between the training progress and β. In particular, the main innovation of this study is as follows:

1. A novel Self-Attention variant termed S2AN is proposed, which quantifies training progress by calculating the similarity of sample distribution in experience pool and establish-es a nonlinear relationship between this progress and the strength of error compensation.
2. A novel parallel sampling approach, called Double-Sampling Mechanism (DSM), is proposed to facilitate the concurrent execution of training and similarity calculations. DSM operates two sampling procedures simultaneous-sly. The PS supplies experience samples for model training, whereas the RUS delivers data inputs for the attention module. Additionally, a mirror buffer that maintains the same data distribution as the original experience pool is constructed to ensure the two sampling procedures are operated separately.

3 A multi-USV pursuit-evasion game scenario is designed. Besides, to show the reason why PER causes extra error, the difference between shifted and original distribution is visualized.

The structure of the rest paper is as follows: Section **3** is the problem statement and the introduction of the basic knowledge. Section **4** describes the design process of ALAP algorithm in detail. Section **5** provides comparative experimental results and analysis. Finally, Section **6** draws the conclusions.

3. Preliminaries

3.1. Markov Decision Process

A Markov model with five key elements $\langle S, A, S', r, \gamma \rangle$ is defined to describe the interaction process between the agent and environment. Among them, the state S represents the possibility configuration of the agent, while vector A denotes the action space. At each time step t, the agent follows the strategy π_θ to select the action A_t, and obtains the next state S'. according to the state transition equation. In this process, the agent receives immediate reward r_t according to a function of its action and state. We call the process of progression from the initial state to the terminated state as a trajectory ζ, and the agent's goal is to maximize the expected return $R = \sum_{t=0}^{T} \gamma_t r_t$ to continuously optimize ζ, where γ and T represent the discount factor and time horizon, respectively.

3.2. Prioritized Experience Replay

PER applies a priority based non-uniform sampling strategy, which adjusts the sampled frequency of experience transitions depending on the magnitude of TD-error $|\delta|$.

It contributes in improvement of the conventional experience replay in two main ways [12]. Firstly, PER assigns each transition i with a sampling probability proportional to its TD error, and improves the training efficiency by learning the important one more frequently as follows:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}. \tag{1}$$

In Equation (1), $p_i = |\delta(i)| + \epsilon$ is the priority of transition i. The hyperparameter α is used to smooth out extremes, and a minimal positive constant value U makes sure that the sampled probability is greater than 0, which gives all transitions a chance to be selected.

Secondly, it introduces importance sampling weight ratios $\overline{w}(i)$ to correct the estimation deviation caused by shifted state-action distribution:

$$\overline{w}(i) = \frac{w(i)}{\max_j w(j)}, w(i) = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{2}$$

$$L_{PER}(\delta(i)) = \overline{w}(i) L(\delta(i)) \tag{3}$$

Note that $\overline{w}(i)\delta(i)$ will be used instead of $\delta(i)$ when updating Equation (3). The hyperparameter β increases linearly from β_0 to 1 throughout the training process.

3.3. Loss Adjusted Prioritized Experienced Replay

The linear annealing of importance weight in PER cannot completely eliminate the deviation, and the sensitivity to outliers is easy to further magnify it. Fujimoto et al. [4] proposed an LAP algorithm, which uses the segmented loss function

$$L_{Huber}(\delta(i)) = \begin{cases} 0.5\delta(i)^2 & |\delta(i)| \le 1, \\ |\delta(i)| & otherwise, \end{cases} \tag{4}$$

combined with priority clipping scheme to further reduce the deviation:

$$P(i) = \frac{max(|\delta(i)|^\alpha, 1)}{\sum_j max(|\delta(j)|^\alpha, 1)}. \tag{5}$$

By combining Equations (4)-(5), it can be seen that when the absolute value of TD error is less than and equal to 1, MSE is applied as the loss function according to Equation (4), and the priority of transition i is cut up to 1, so that the uniform sampling is performed for each transition i with a sampled probability of $1/N$ according to Equation (5). Similarly, if the TD error is greater than 1, MAE is used to suppress the sensitivity to outliers in (4), and the

non-uniform sampling is performed according to Equation (5). Although LAP avoids the interference of outliers during training, the uniform sampling part decreases the convergence rate. Furthermore, MAE can only suppress the sensitivity to outliers, but cannot correct the shifted distribution, and thus, the deviation does exist.

3.4. Multi-agent Twin Delayed Deep Deterministic Policy Gradient

MATD3 is an expanded form of TD3 for multi-agent environments, which addresses the overestimation problem by applying twin target network. MATD3 follows the centralized training and distributed execution paradigm utilized by MADDPG [15], the training target with centralized critics for each agent are shown as follows:

$$y_i = r_i + min_{j=1,2} Q^{\pi}_{i,\theta_j}(x', a'_1, ..., a'_N), \quad (6)$$

where x' and $a'_i,...,a'_N$ represent the full state information and the joint action at the next time step, and r_i is the immediate reward that agent i receive. The minimum value of the two Q network is applied when y_i is being computed, which suppresses the overestimated Q function posed by the maximized operator.

4. Attention Loss Adjusted Prioritized Experience Replay

In this section, the design of ALAP algorithm is described in detail from two parts: Shuffling Self-Attention network and Double-Sampling mechanism.

4.1. Design of Shuffling Self-Attention Network

In PER and its improved variants, the hyperparameter β is an important index to regulate the importance sampling weight (IS), which determines the correction strength with respect to error caused by PER. During training, PER employs the linear annealing method to make β grow linearly with the training episodes increase. This implies that β is positively correlated with the training progress Ω: $\beta \propto \Omega$ [21, 32]. In ideal case, β reaches 1 at the same time as the model converges to fully compensate the estimation error. However, the training progress is not uniformly distributed over the scale of episodes number. Training will most likely not end when the model converges, and this linear parameter tuning may exacerbate the error due to the inability to accurately capture the correspondence between the training progress and the number of training episodes. In summary, the episodes number alone cannot be relied upon to estimate the training progress accurately.

In order to accurately quantify the training progress, we build an improved Self-Attention network called S2AN to measure the training progress by calculating the similarity of sample distribution in the experience pool. At the beginning of training, the transitions in experience pool generated by the stochastic exploration of agent are random and uniform. As the training proceeds, the agent will make more use of the current optimal strategy to select high-return actions. In other words, given a state, the action selected by the agent will be considerably similar, and the similarity of transitions sequence generated by the interaction between agent and environment will be improved. Obviously, there is a positive correlation between sample similarity and training progress as well: $\Omega \propto \Phi$, where Φ represents the sample similarity. Consequently, On the basis of: $\beta \propto \Omega$ and $\Omega \propto \Phi$, we yield:

$$\beta \propto \Omega \propto \Phi. \quad (7)$$

We input the mini-batch sized state-action pair sequences $X = [(s_1, a_1), (s_2, a_2), ..., (s_n, a_n)]$ into S2AN, whose output is the quantified value of the sample similarity in experience pool, and β is obtained after normalization according to Equation (7).

The attention function can usually be described as a nonlinear fitting network that maps a query and a set of key-value pairs to the output [10]. Its output is essentially a weighted sum containing importance weights. In practice, we package queries, keys and values into the corresponding matrices Q, K and V, and the output of attention value is calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}})V, \quad (8)$$

where d_k is the dimension of queries and keys vectors, dividing the dot-product $Q \cdot K^T$ by $\sqrt{d_k}$ to prevent the gradient from disappearing.

To meet the requirements for similarity calculations concerning the internal elements of the input sequence, S2AN optimizes the Self-Attention network properly and removes the value vector. The attention module only outputs the similarity of key-query pairs. Considering that X is a list constructed of n state-action pair vectors. Referring to the physical meaning of vector projection, we measure the similarity between Q and K by the sum of the projections among the corresponding vectors, where $Q = K_0 = XW_Q$, $K = shuffle(K_0)$, and both of them maintain dimensions consistent with X. W_Q represents the initial weight matrix of Q. Since the similarity is calculated for the internal elements of X, we must perform projection operation for each q_i and k_i after randomly rearranging the order of the elements (by shuffle operation) in Q. In fact, K is identical to Q before the shuffle operation. Assuming that the elements inside Q are all highly similar, Q and K are similar as well no matter how one disrupts the ordering of the elements inside Q to obtain K. On the contrary, the query-key pair will show great difference after shuffle operation if the elements inside Q are quite different in the beginning. The process for S2AN to calculate the attention value is shown as follows:

$$S2AN(Q,K) = softmax(\frac{Q \bullet K^T}{\sqrt{d_k}}) \tag{9}$$

$$Q \bullet K^T = \sum_{i=1}^{m} \frac{(Q_i \cdot K_i)}{|K_i|} \tag{10}$$

where $\cdot$ stands for projection sum operation of the corresponding elements between Q and K, Q_i and K_i denote the elements at the position of index i in the Q and K sequences, respectively.

To clarify our motivation further, a specific example is provided to describe the shuffle operation in mathematical language. Since K is identical to Q at the initial stage of query-key process, we have: $K_0 = Q$, where K_0 is the initial value of K. Suppose that the input sequence X of S2AN has a length n of 5. Then, we got $K_0 = Q = [q_1,...,q_j,...,q_5]$, where q_j stands for the element in Q with index j. As a result, the expansion of $Q \cdot K_0$ can be written as:

$$\begin{aligned} Q \bullet K_0^T &= \sum_{i=1}^{5} \frac{(Q_i \cdot K_i)}{|K_i|} = \frac{(Q_1 \cdot K_1)}{|K_1|} + \frac{(Q_2 \cdot K_2)}{|K_2|} \\ &+ \frac{(Q_3 \cdot K_3)}{|K_3|} + \frac{(Q_4 \cdot K_4)}{|K_4|} + \frac{(Q_5 \cdot K_5)}{|K_5|} \\ &= \frac{(q_1 \cdot q_1)}{|q_1|} + \frac{(q_2 \cdot q_2)}{|q_2|} + \frac{(q_3 \cdot q_3)}{|q_3|} + \frac{(q_4 \cdot q_4)}{|q_4|} + \frac{(q_5 \cdot q_5)}{|q_5|} \end{aligned} \tag{11}$$

and it can be understood more intuitively by referring to Figure 2(a). Since $K_0 = Q$, we let $Q \cdot K_0^T$ represent the maximum value of similarity between Q and K. Subsequently, we obtain K after operating shuffle operation on K_0 : $K = shuffle(K_0)$. For ease of comparison, we expand $Q \cdot K$ as well:

$$\begin{aligned} Q \bullet K^T &= \sum_{i=1}^{5} \frac{(Q_i \cdot K_i)}{|K_i|} = \frac{(Q_1 \cdot K_1)}{|K_1|} + \frac{(Q_2 \cdot K_2)}{|K_2|} \\ &+ \frac{(Q_3 \cdot K_3)}{|K_3|} + \frac{(Q_4 \cdot K_4)}{|K_4|} + \frac{(Q_5 \cdot K_5)}{|K_5|} \\ &= \frac{(q_1 \cdot q_3)}{|q_3|} + \frac{(q_2 \cdot q_2)}{|q_2|} + \frac{(q_3 \cdot q_5)}{|q_5|} + \frac{(q_4 \cdot q_1)}{|q_1|} + \frac{(q_5 \cdot q_4)}{|q_4|} \end{aligned} \tag{12}$$

and it is shown in Figure 2(b). It is important to note that K_i acts more like a pointer, the variable q_j to which the pointer points may change, but the pointer itself remains fixed for both K_0 and K.

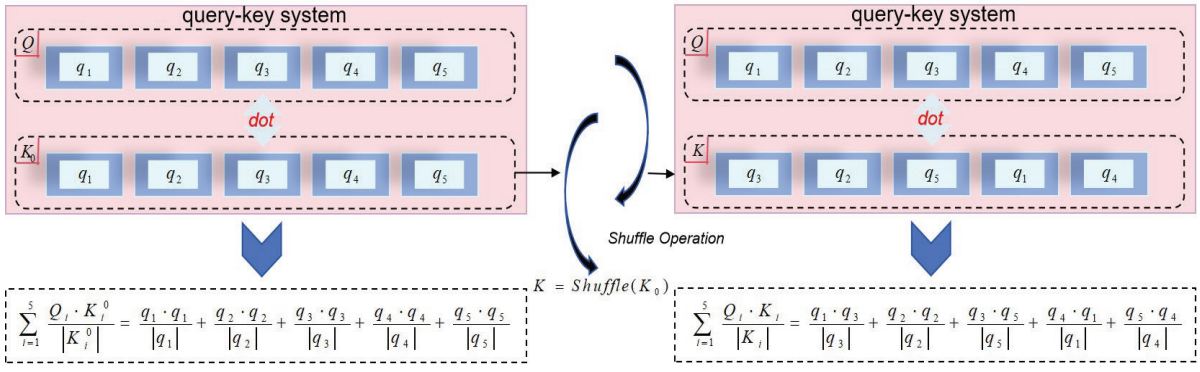
After the shuffle operation, due to the random sampling input of S2AN, there will be a significant disparity between q_j during the early stages of training, resulting in a notable difference between $Q \cdot K_0^T$ and $Q \cdot K^T$. As training proceeds, the similarity among q_j will increase, making $Q \cdot K^T$ converge towards $Q \cdot K_0^T$.

Specifically, as the model approaches convergence, the training progress Φ nears its maximum value Φ^*. At this point, we have: $q_1 \approx q_2 \approx ... \approx q_5 \approx (s_*, a_*)$, where (s_*, a_*) represents the optimal transition that can access the highest reward. Moreover, since $\beta \infty \Omega \infty \Phi$ according to Equation (7), β and Ω will reach the maximum value simultaneously as $\Phi \to \Phi^*$. In such cases, we yield:

$$\lim_{\substack{\beta \to 1 \\ \Omega \to \Omega^* \\ \Phi \to \Phi^*}} Q \bullet K^T \to Q \bullet K_0^T \tag{13}$$

where Ω^* is the maximum value of Ω.

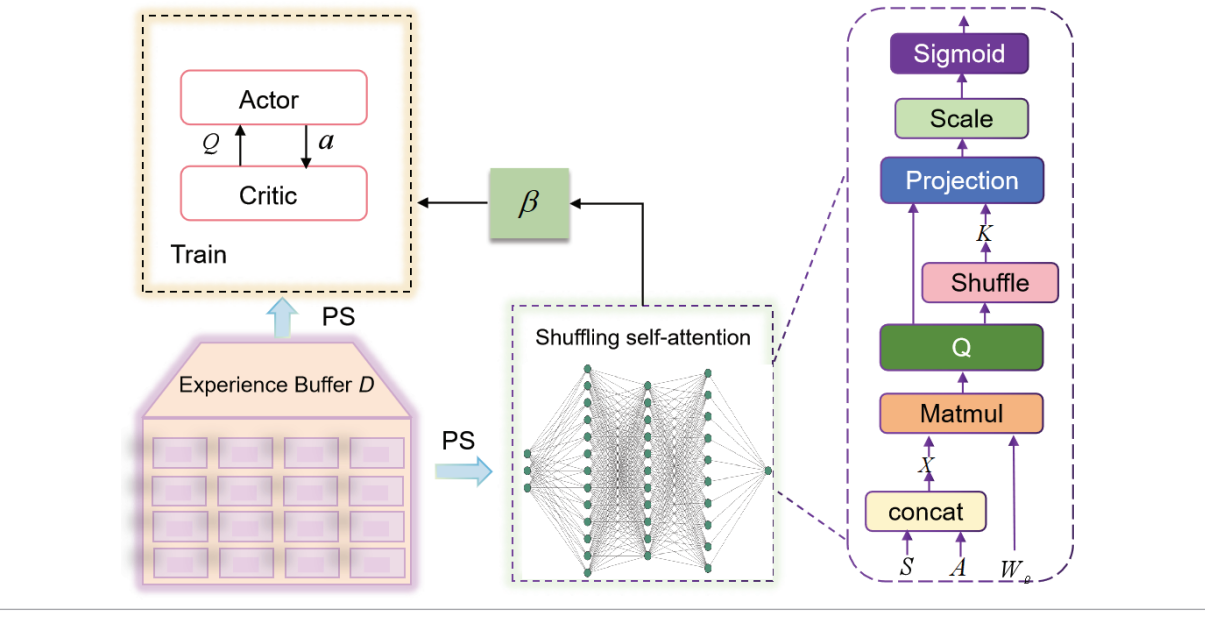
Figure 2
Shuffle operation.



(a) query-key system without shuffle operation (b) query-key system with shuffle operation

The input of the S2AN is the state-action pair sequence, and the output attention value represents the similarity among the elements in X. After passing through the fully connection layer, the attention value is normalized by the activation function to obtain β. The schematic diagram of Shuffling Self-Attention mechanism is shown in Figure. 3. Note that the schematic diagram in this paper only depicts the network structure of ALAP under A-C framework, and the combination of ALAP with the algorithm based on value-function is similar and it is omitted here. We can see that the Attention network's input is a mini-batch sized sequence of experience transitions provided by PS in Figure 3. However, PS exploits the fixed standard to screen out samples, thus its sample distribution cannot restore the real situation in experience pool, which inspires us to design the Double-Sampling mechanism.

Figure 3
Shuffling Self-Attention network.



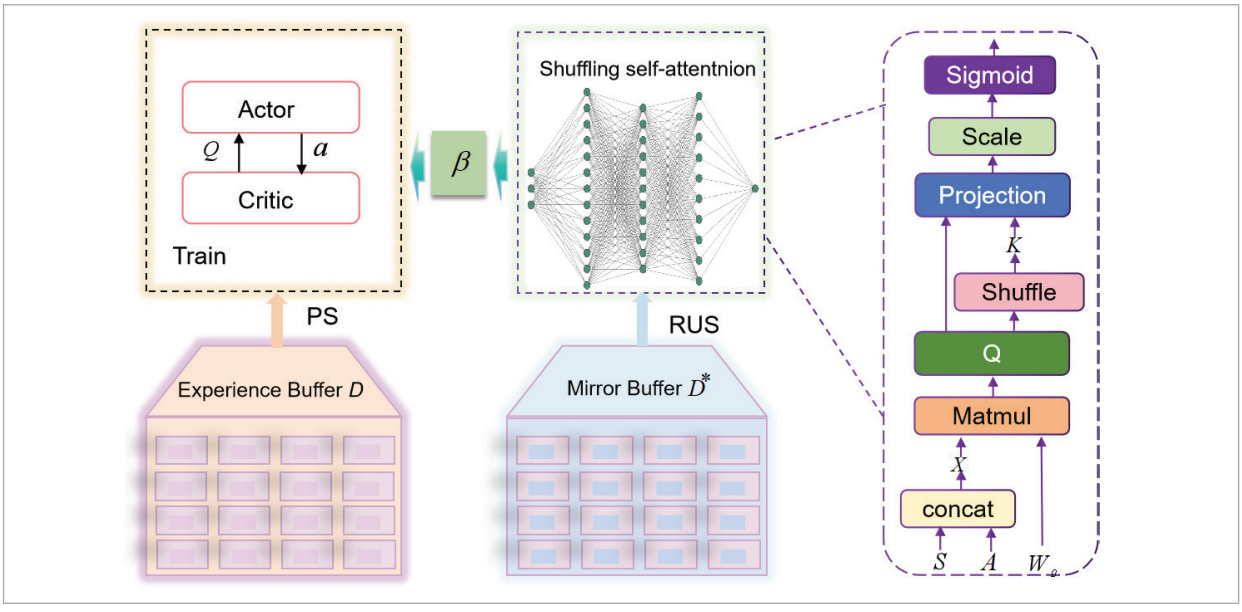
4.1. Design of Double-Sampling Mechanism

The core idea of ALAP is to quantitatively describe the training progress by calculating the similarity of the sample distribution in experience pool through the attention module. Therefore, the sample distribution from data source of the attention module should be consistent with the sample distribution in experience pool. However, the priority-based sampling way artificially changes sample's visited frequency, so that the data source obtained by PS cannot reflect the real situation of data distribution in entire experience pool. Hence, we need to rely on random uniform sampling to provide data input for the attention module. In addition, the model network still needs PS to provide high-quality training samples. To sum up, we propose a parallel sampling method called Double-Sampling mechanism, which incorporates both PS and RUS. For ease of description, we define the data source acquired by PS as s and the data source obtained by RUS as s^*.

During the training process, the priority-based sampling provides data samples for training model, and the uniform sampling is only responsible for providing data as input for attention module to fit β, allowing real-time correction of estimation error. The key purpose of the Double-Sampling mechanism is not to interfere with the normal operation of training, and to collect data in the same experience pool D using both sampling procedures simultaneously. Since the two sampling procedures are performed simultaneously, even though the sampled batch size is much smaller than the volume of the experience pool D, there still remains certain probability that the same sample will be selected for both sampling methods, resulting in instability of the algorithm. To overcome this, we propose a concept of mirror buffer D^* and construct a mirror experience pool with the same data distribution as the original one. At each iteration step, the mirror buffer D^* adds and updates experience samples at the same time as the original experience pool D to keep the data synchronized. It is worth noting that, although from the physical level, the two sampling steps are carried out in two different buffers, but from the data level, they are actually operated in the same experience pool. The schematic diagram of Double-Sampling mechanism is shown in Figure 4.

Finally, in order to reduce the sensitivity of the algorithm to outliers, ALAP follows the Huber loss function applied in LAP. However, unlike LAP, we remove the sample priority clipping component and use PER-sampled data to train the model throughout the whole process to ensure unbiased Q-value function estimation without compromising convergence rate. The overall operation flow of ALAP is shown in **Algorithm 1.**

Figure 4
Double-Sampling mechanism.



Algorithm 1 ALAP Algorithm

Input: mini-batch n, step-size σ, replay period K, buffer storage N, exponent α and β, budget T, tiny positive constant U.
1: Initialize replay buffer $D = \varnothing$, mirror replay buffer $D^* = \varnothing$, $p_1 = 1$, $\Delta = 0$
2: Observe environment state s_0 and choose action a_0
3: **for** $t = 1$ **to** T **do**
4: Observe s_t, r_t, γ_t
5: Store transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in D
6: Store transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in D^*
7: **if** $t > K$ **then**
8: **for** $j = 1$ **to** K **do**
9: Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
10: Compute TD-error δ_j and update transition priority $p_j \leftarrow |\delta(j)| + \epsilon$ 11:
11: Obtain transitions s with mini-batch n from D by PS for model training
12: Obtain transitions s^* with mini-batch n from D^* by RUS
13: Compute β through Shuffling Self-Attention network according to s^*
14: Compute importance sampling weight $\overline{w}(i) = (N \cdot P(j))^{-\beta} / \max_i w(i)$
15: Compute weight-change $\Delta \leftarrow \Delta + \overline{w}(j) \cdot \delta(j) \cdot \nabla_\theta Q(s_{t-1}, a_{t-1})$
16: **end for**
17: Update weights $\theta \leftarrow \theta + \sigma \cdot \Delta$, reset $\Delta = 0$
18: Copy parameters to target network $\theta_{target} \leftarrow \theta$
19: **end if**
20: Choose action $a(t) \sim \pi_\theta(s_t)$
21: **end for**

5. Experiment

The comparative experiments are carried out in **Atari 2600** benchmark and **multi-USV pursuit-evasion game**, employing DQN and MATD3 as baseline, respectively. To assess the effectiveness of ALAP, 15 games from the Atari 2600 benchmark are selected for testing. To further exhibit ALAP's versatility, we integrate it with MATD3 and evaluate its performance in the 3v3 USV pursuit-evasion game. We reveal the root reason of estimation errors in Q function caused by PER, and visualize the difference between the shifted and original state-action distribution under different batch size configurations. In each environment, several groups of comparative experiments are carried out, we use the same network structure, reward shaping and hyperparameter configuration to train the algorithms for comparison. The primary distinction between the improved algorithm and the baseline is mainly reflected in the different sampling ways and error compensation strategies.

5.1. Results in Atari

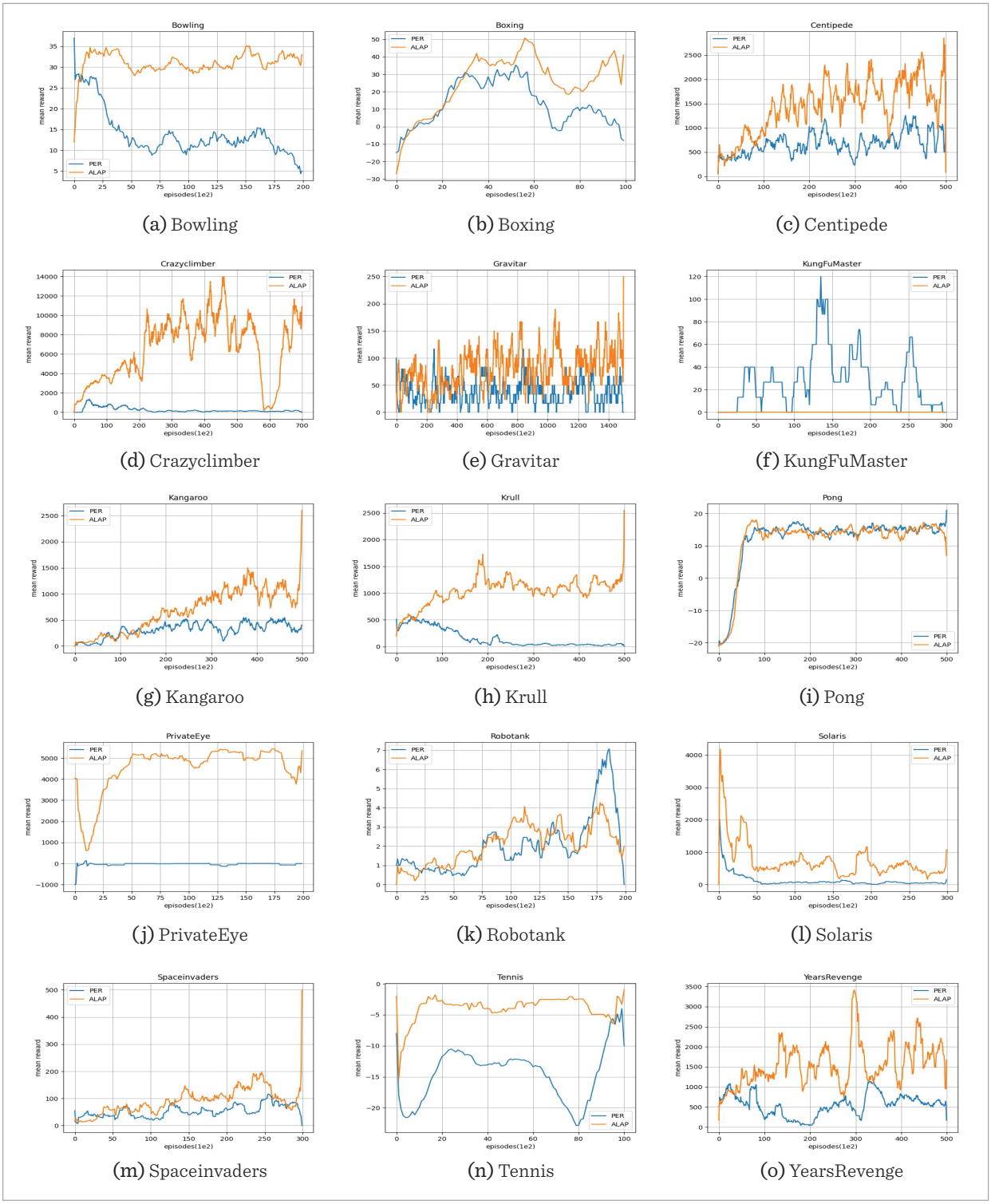
In Atari 2600 benchmark, we follow the original network structure and hyperparameters setting of PER. We applied Adam optimizer to update model according to transitions screen out from buffer. The batch size is 32, while the learning rate and discount factor are 0.0001 and 0.99, respectively. Each game is run with the same random seed (seed=1) initialization. The action choosing by agent depends on Ú–*greedy* strategy with U decaying from 1 to 0.02.

Figure 5 shows the reward comparison curves in 15 different games. Moreover, an evaluation summary of the results presented in Figure 5 is provided in Table 1 to improve readability.

To intuitively visualize the increasing performance of ALAP relative to PER, we define the normalized score as follows:

Figure 5

Atari 2600 results. This graph presents a comparison of the reward curves for ALAP and the baseline PER across 15 different Atari games.



(a) Bowling (b) Boxing (c) Centipede

(d) Crazyclimber (e) Gravitar (f) KungFuMaster

(g) Kangaroo (h) Krull (i) Pong

(j) PrivateEye (k) Robotank (l) Solaris

(m) Spaceinvaders (n) Tennis (o) YearsRevenge

$$normalized\ score = \frac{maxALAP_{score} - maxPER_{score}}{abs(maxPER_{score})}, \quad (14)$$

where the $ALAP_{score}$ and the PER_{score} represent the scores obtained by the corresponding algorithm. The normalized score between ALAP and PER is shown in Figure 6.

Figure 6 depicts the performance improvement of ALAP relative to PER in terms of percentage. 0% on the y axis indicates equivalent performance. Positive values represent the case where ALAP performs better, and vice versa. In conclusion, ALAP demonstrates superior performance compared to PER in most of games.

Table 1

Mean return and standard deviations (STD) across 15 different games, where bold values represent the best-performing under each environment.

Game	PER		ALAP	
	mean value	std	mean value	std
Bowling	13.93	9.36	**31.13**	**7.08**
Boxing	13.21	28.55	**25.42**	**26.42**
Centipede	736.61	928.98	**793.01**	**891.38**
Crazyclimber	247.57	**408.63**	**6508.08**	5399.05
Gravitar	37.3	110.65	**80.2**	**83.43**
KungFuMaster	27.33	77.37	**0.0**	**0.0**
Kangaroo	309.8	**297.39**	**647.8**	684.1
Krull	154.72	**213.46**	**1039.41**	500.43
Pong	0.98	15.05	**1.21**	**13.54**
PrivateEye	-24.94	**358.66**	**4507.24**	1634.65
Robotank	2.07	2.74	**2.50**	**2.12**
Solaris	121.60	**308.73**	**736.53**	1305.91
Spaceinvaders	53.63	98.94	**89.00**	**65.52**
Tennis	-14.54	6.67	**-3.93**	**5.53**
YearsRevenge	554.24	**564.74**	**1554.49**	1416.12

Figure 6

Normalized score.

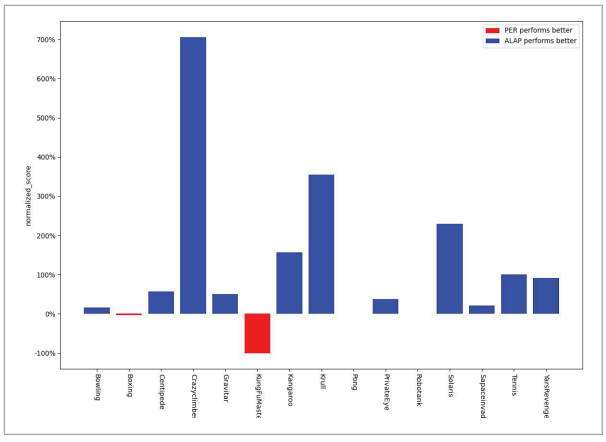
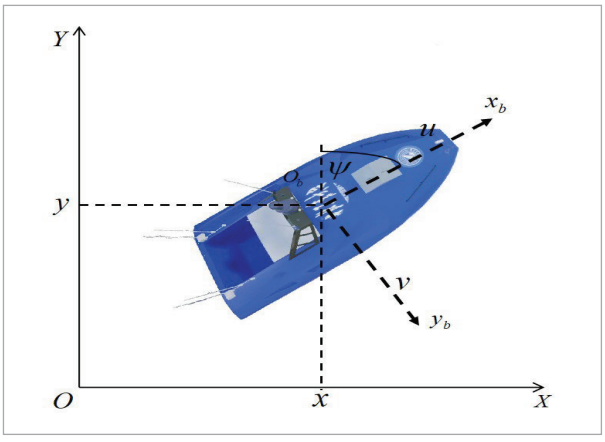


Figure 7

The kinematic model of USV.



5.2. Results in Multi-USV Game

We discuss a multi-USV pursuit-evasion game in this subsection. The kinematic model for each USV is shown below:

$$\begin{cases} \dot{x}_i = & u_i cos\psi - v_i sin\psi_i, \\ \dot{y}_i = & u_i sin\psi + v_i cos\psi_i, \\ \psi_i = & arctan(\dfrac{v_i}{u_i}), \end{cases} \tag{15}$$

where $i \in (p,e)$, $p = (p_1, p_2, p_3)$ and $e = (e_1, e_2, e_3)$ are 3 pursuers and 3 evaders, respectively.

Equation (15) can be understood more intuitively by referring to Figure 7. We place the USV i in the geodetic coordinate system OXY, where tuple (x_i, x_y) is the coordinate orientation of i. ψ_i is the yaw angle between USV's hull longitudinal axis x_b and the Y axis. (u_i, v_i) represents the velocity vector of i, and the ψ_i can be easily calculated through the trigonometric relationship. The control inputs to the USV are the longitudinal and transverse acceleration vector (a_u, a_v) with $u_{min} < a_u{*}t < u_{max}$, $v_{min} < a_v{*}t < v_{max}$. $(u_{min}, u_{max}, v_{min}, v_{max})$ are maximum and minimum values of (u_i, v_i). So far, we have transformed the multi-USV pursuit-evasion game into a reinforcement learning problem with state space (x_i, y_i, ψ_i) and action space (a_u, a_v).

We place the multi-USV system in a specific sea area surrounded by walls. The obstacle avoidance problem is considered, and two obstacles (b_1, b_2) are randomly added in the area. Consequently, the locations of USV and obstacle need to satisfy the constraints:

$$\begin{cases} x_{min} \le x_i \le x_{max}, \\ y_{min} \le y_i \le y_{max}, \end{cases} \tag{16}$$

where $(x_{min}, x_{max}, y_{min}, y_{max})$ are maximum and minimum coordinate values of the specific sea area. During the game, the pursuer must approach the evader closely enough to bring the evader within the fire range, that is $r_c \le d_{attacking}$. r_c is the fire range of pursuer and $d_{attacking}$ represents the relative distance between pursuer and evader.

If the pursuer fulfill the capture requirement, the evader will be considered "sunk" and it will be removed from the area. When that occurs, the pursuer will get a positive reward R_a immediately, and the evader will receive an exact opposite negative reward $-R_a$. In addition, the activities of USV are strictly limited to the wall or they will receive a penalty R_b. To realize the obstacle avoidance function, we integrate a collision penalty mechanism into the system, where the USV colliding with an obstacle or a companion will receive a punishment from R_c. To summarize, the reward of each USV can be written as $R = R_a + R_b + R_c$.

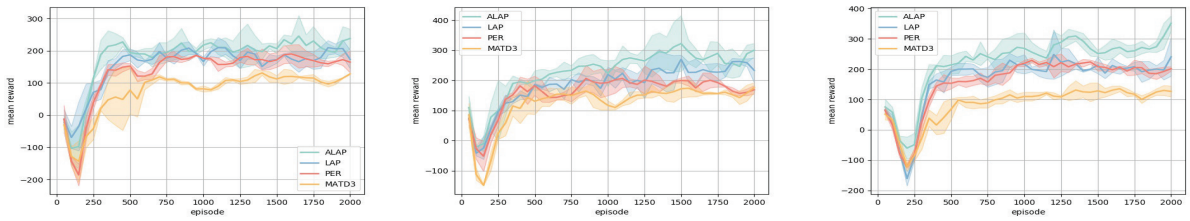
In the comparison experiments, all algorithms execute different sampling manners to draw mini-batch samples from the corresponding experience pool of volume 10^6 for model training, which is constructed by 2-layer ReLU MLP with 64 units per layer. The total number of training episodes is 2000, and the maximum simulation step is 25. The Adam optimizer is used to optimize the network parameters with a learning rate of 0.001 and a discount factor of 0.95. To avoid unintended outcomes, we investigate the effectiveness of ALAP against other algorithms under batch size=32, 64, 128. 10 independent runs are carried out in each case with different random seeds initialization from 11 to 20. To compare the algorithm variance and stability, we set a 50 confidence interval, and draw the confidence bands of each curve with shaded region, which is applied in all batch size configuration. For ease of description, we refer algorithm X + MATD3 as algorithm X. The simulation results of multi-USV game are shown as follows:

Figure. 8 shows the average reward curves of ALAP, LAP, PER, and the conventional MATD3. It demonstrates that ALAP has the best performance with respect to convergence speed and mean reward value in the face of different batch-size configurations. Similar with Atari benchmark, an evaluation summary of the results presented in Figure 8 is provided in Table 2.

To further assess the stability of the algorithm throughout the training phase, the variance curves are drawn in Figure 9, which shows the inhibitory effect of ALAP on the estimation error.

Figure 8

Mean reward of all algorithms. The curves of (a), (b) and (c) describe the average reward over 10 trials under various batch-size configuration, respectively, with the shaded area representing the 50% confidence interval.



(a) batch-size=32 (b) batch-size=64 (c) batch-size=128

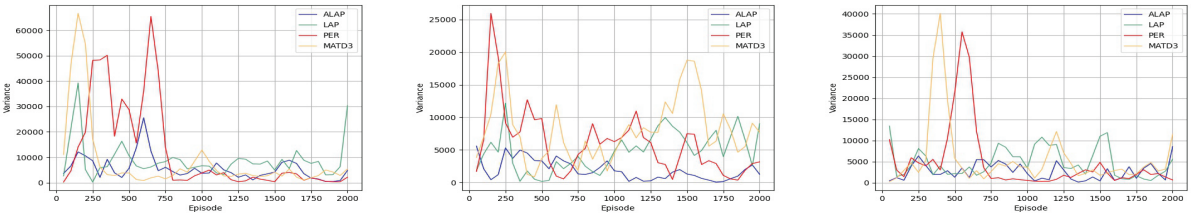
Table 2

Mean return and standard deviations (STD) across different batch-size, where bold values represent the best-performing under each environment.

Batch-size	MATD3		PER		LAP		ALAP	
	mean	std	mean	std	mean	std	mean	std
32	75.52	**66.27**	134.09	84.37	156.32	67.44	**181.13**	71.78
64	122.31	73.21	158.78	78.93	184.63	70.54	**226.86**	**59.75**
128	84.29	95.55	158.03	86.86	165.93	90.35	**222.14**	**80.19**

Figure 9

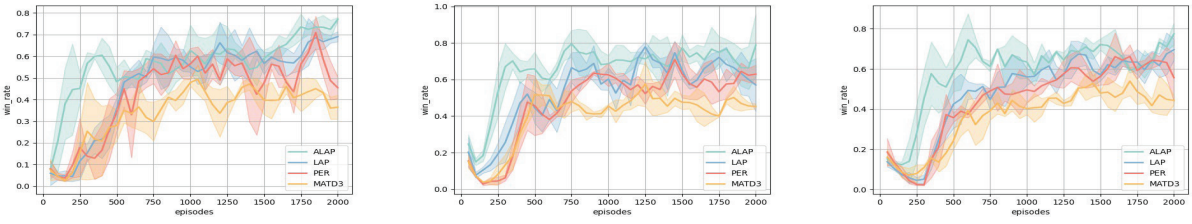
Variance curves describe the stability of the entire training phase.



(a) batch-size=32 (b) batch-size=64 (c) batch-size=128

Figure 10

Win rate of all algorithms. The curves of (a), (b) and (c) describe the average win rate over 10 trials under various batchsize configuration, respectively, with the shaded area representing the 50% confidence interval.



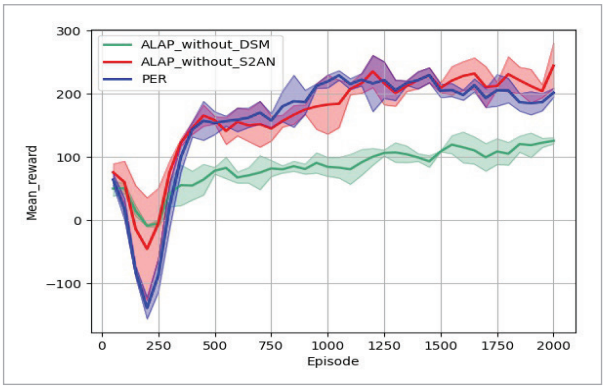
(a) batch-size=32 (b) batch-size=64 (c) batch-size=128

During game, if all evaders are captured by the pursuers within the specified time limit, the pursuers are deemed as winners. Every 50 rounds, we tally the number of wins Λ for pursuers, and denote their win rate as $\frac{\Lambda}{50}$. As a result, Figure 10 illustrates the win rate curves of pursuers controlled by the baseline and different PER variants. Similar with Figure 8, ALAP presents superior performance compared to other training frameworks. It maintains a consistent win rate exceeding 70% across all batch size configurations, and requires only 250 training rounds to achieve 60% .

Since ALAP contains two main components, the ablation study is provided to separately analyze the individual contribution of S2AN and DSM.

Figure 11

Ablation study.



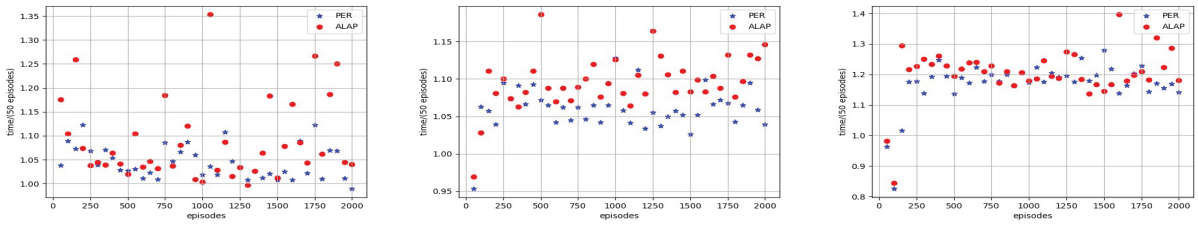
As illustrated in Figure 8, ALAP significantly outperforms PER. However, upon examining Figure 11, we observe that the performance of ALAP declines dramatically when it loses either S2AN or DSM. From the aspect of S2AN, it enables PER to have the ability to construct the nonlinear mapping relationship between β and training progress, If ALAP is not equipped with S2AN, it has no difference with conventional PER. Switching to DSM's aspect, it is an essential technical means of ALAP, which is responsible for satisfying the diverse data requirements for both S2AN and model training. If DSM is removed from ALAP, the data input of S2AN is solely dependent on PS, which makes the data distribution of S2AN unable to simulate the original situation in experience pool, resulting in a huge estimation deviation. As a result, both S2AN and DSM are indispensable for ALAP.

The computational cost of ALAP is discussed from its two functional module: S2AN and DSM. We assume that the input sequence length for the neural network is n, determined by the mini-batch size, with each input element having a feature dimension of d. The conventional SAN contains three main components including similarity calculation, softmax, and weighted summation. The shape of Q and K are (n, d) and (d, n), respectively. In such case, the similarity calculation between Q and K generate complexity $O(n^2 \cdot d)$. Subsequently, the softmax is operated on the $(n \times n)$ matrix generated by $Q \cdot K$, and this will bring computation cost of $O(n^2)$. Similarly, the complexity of weighted summation is $O(n^2 \cdot d)$, which is because of the calculation taking place between the weight matrix $(n \times n)$ and V $(n \times d)$. As a result, the time complexity of SAN is obtained: $T(n)_{SAN} = O(n^2 \cdot d) + O(n^2) + O(n^2 \cdot d)$. The computing resource is considered during the design process of S2AN, in such cases, S2AN only remain two components including similarity calculation and softmax, and the time complexity of S2AN can be written as: $T(n)_{S2AN} = O(n^2 \cdot d) + O(n^2)$.

Consequently, since $T(n)_{S2AN} < T(n)_{SAN}$, S2AN is more economical and reliable than SAN in terms of computing overhead.

Similarly, from the complexity of DSM, it can be decoupled into two components: PS and RUS. Specifically, we assume that the volume of experience buffer is N, and the batch-size of mini-batch is 1. The random uniform sampling (RUS) can directly sample transitions from buffer with computational cost $O(1)$. Compared to RUS, the priority-based sampling (PS) need to update the sample prioritization and select samples according to their labeled priority order. The computational complexity is $O(\log N)$ due to the utilization of a sum tree data structure. In this case, the time cost of DSM can be yield: $T(n)_{DSM} = O(\log N) + O(1)$. For conventional PER, it solely depends on PS for sampling, and its time cost is: $T(n)_{PER} = O(\log N)$. Generally speaking, the value of N is much greater than 1. Therefore, we have: $T(n)_{DSM} \approx T(n)_{PER}$. In summary, on the basis of $T(n)_{S2AN} < T(n)_{SAN}$ and $T(n)_{DSM} \approx T(n)_{PER}$, we demonstrate that the algorithm we proposed conserves computing resources more efficiently than the existing methods. To further substantiate our perspective, we present Figure 12 that illustrates the time cost comparison between ALAP and the baseline.

Figure 12
Training times comparison between ALAP and the baseline PER.



(a) batch-size=32 (b) batch-size=64 (c) batch-size=128

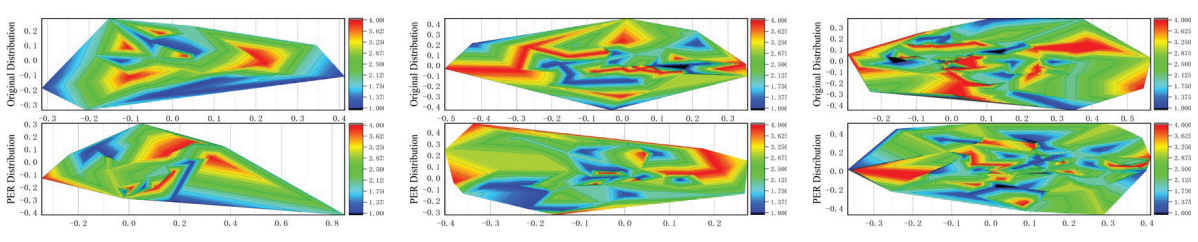
Figure 12 illustrates the disparity in training duration between ALAP and PER. We evaluate the training time at intervals of every 50 rounds. The maximum discrepancy between ALAP and PER remains within 0.3 seconds. Although the time expenditure for ALAP is slightly greater than that of PER, the difference is still considered acceptable.

As described in **Algorithm 1**, off-policy reinforcement learning selects n samples from the experience pool by different sampling strategies (PS, RUS) each round to form the distribution s and s^*, which are used to train the model and estimate the Q-function. The underlying reason of the estimation error generated by PER is that PS shift the distribution s^* to s. In order to demonstrate this phenomenon intuitively, we visualize s and s^* through a contour heat map. We describe the state in terms of the distribution of pixel points and visualize the action with different colors. It is worth noting that the output of the actor network is an action probability distribution, and for ease of description, we express the action selection process in the form of one-hot encoding. We disassemble the action space of USV into 4-dimensional discrete variables, $[-u, u, -v, v]$, where u and v are the basic units for the accelerated change of the USV vector, corresponding to channels [1,2,3,4], respectively. To facilitate visual display, we only activate the channel obtains the maximum probability during each action selection. As illustrated in Figure 13, there is a significant disparity between s and s^* across various batch-sizes. Therefore, it is essential to implement precise compensation for PER with an adjustable parameter β.

Figure 14 shows the simulation scenario of a 3v3 USV pursuit-evasion game in Unreal Engine. As can be seen in subfigure (a), the 6 USVs are divided into two camps: yellow and blue, and randomly initialized at any location of the map. Each USV has a fire-control irradiation radar deployed in front of it, which behaves as a fan shape consisting of several red lines. Once the evader is in radar range, it will be sunk, and the pursuer will look for the next target.

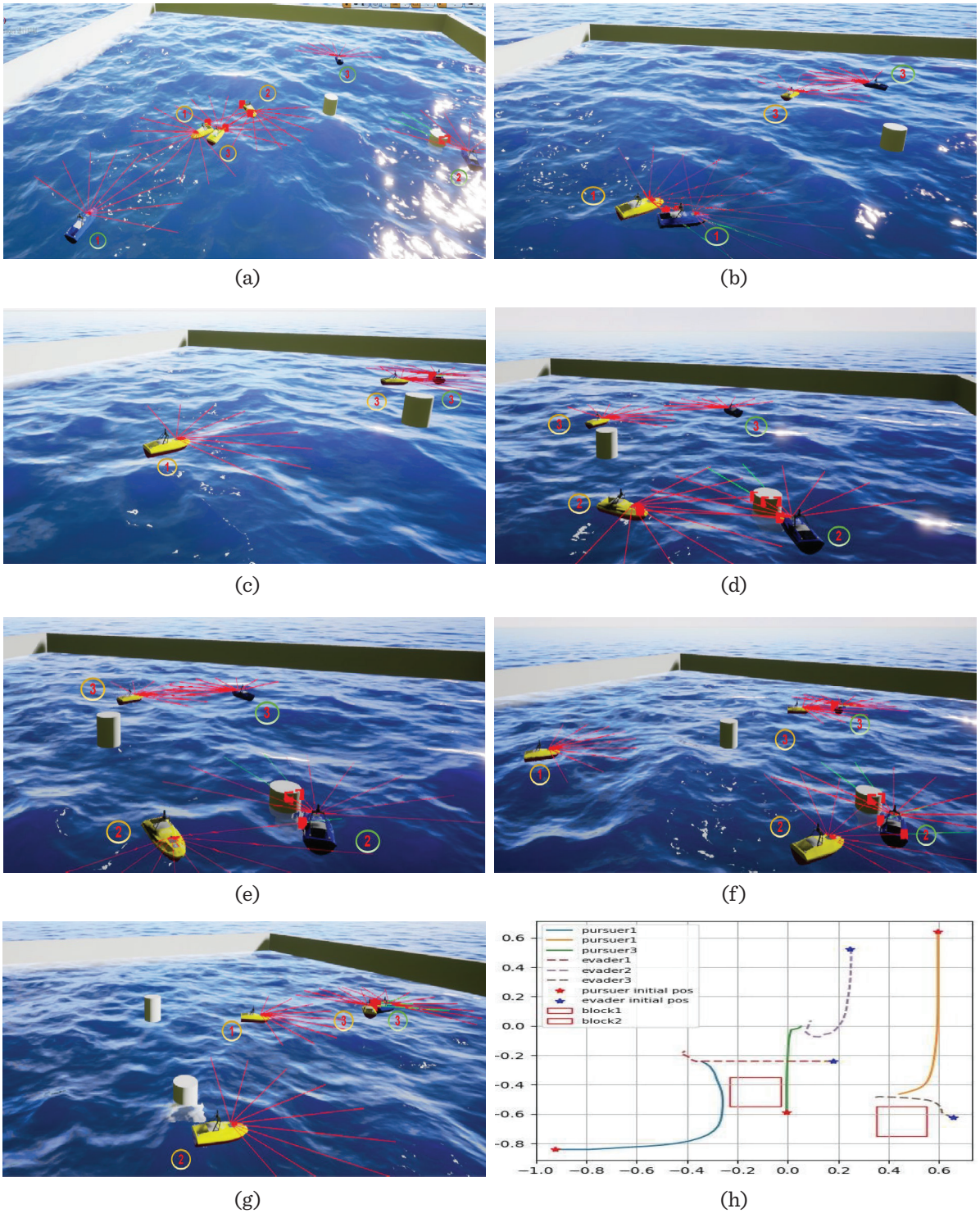
In subfigure (b), the 1st USV of yellow has caught up with the corresponding one in blue camps. Moving to subfigure (c), the blue ship has already sunk. Switching to subfigure (d), yellow 2 is trying to ap-

Figure 13
The difference between s and s^*.



(a) batch-size=32 (b) batch-size=64 (c) batch-size=128

Figure 14
Multi-USV pursuit-evasion scenario.



proach the blue USV. However, in subfigure (e) we find that the yellow USV adjusts its course to avoid the obstacle, and re-engages with the blue USV in subfigure (f). It's clear in subfigure (g) that blue 1,2 are sunk, and all yellow USVs will chase the remaining blue 3. Subfigure (h) depicts the trajectory of the multi-USV system in the 2D plane.

6. Conclusions

In this paper, a general reinforcement learning algorithm framework called ALAP is proposed, which significantly reduces the estimation deviation of Q-value function caused by PER. Firstly, the loss function is described in segments by Huber equation, which suppresses the sensitivity of the algorithm to outliers. Furthermore, the specific progress of training is quantified by calculating the similarity of samples in experience pool through the improved Self-Attention mechanism, enabling precise fitting the hyperparameter β to regulate the importance sampling weight. In addition, we design a Double-Sampling mechanism based on mirror buffer, utilizing two sampling methods simultaneously to provide data sources for training and attention module to ensure algorithm's stable operation. The comparative results obtained from the Atari 2600 verify that ALAP greatly enhances both the speed and stability of training. Among the 15 Atari games, ALAP defeated PER in 11 of them, especially in Crazyclimber, where it achieved an average reward that was 26 times higher than that of PER. Additionally, the pursuit-evasion game in multi-USV system further demonstrates the practical value of ALAP for unmanned platform.

Disclosure Statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant U24B20183, U22B2039, Grant 62273281, and Grant 61922068, and in part by the Aoxiang Youth Scholar Program under Grant 23GH0202058. This work was partially supported by the Australian Research Council (DE250101335).

Acknowledgement

It is important to acknowledge that the initial draft of this article was published as a preprint on arXiv, titled "Attention Loss Adjusted Prioritized Experience Replay." (https://doi.org/10.48550/arXiv.2309.06684)

References

1. Bing, Z. Solving Robotic Manipulation with Sparse Reward Reinforcement Learning via Graph-Based Diversity and Proximity. IEEE Transactions on Industrial Electronics, 2023, 70(3), 2759-2769. https://doi.org/10.1109/TIE.2022.3172754
2. Chen, Z., Li, H., Wang, Z., Yan, B. Weighted Mean Field Q-Learning for Large Scale Multiagent Systems. IEEE Transactions on Industrial Informatics, 2025. https://doi.org/10.1109/TII.2025.3575139
3. David, A., Nir, F., Ronald, P. Generalized Prioritized Sweeping. Proceedings of the 10th International Conference on Neural Information Processing Systems, 1997, 1001-1007.
4. Fujimoto, S., Meger, D., Precup, D. An Equivalence Between Loss Functions and Non-Uniform Sampling in Experience Replay. Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020, 33, 14219-14230. https://doi.org/10.48550/arXiv.2007.06049
5. Gao, J., Li, X., Liu, W., Zhao, J. Prioritized Experience Replay Method Based on Experience Reward. International Conference on Machine Learning and Intelligent Systems Engineering, 2021, 214-219. https://doi.org/10.1109/MLISE54096.2021.00045
6. Hessel, M., Modayil, J., Hasselt, V. H., Schaul, T. Rainbow: Combining Improvements in Deep Reinforcement Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 2018, 32(1), 3215-3222. https://doi.org/10.1609/aaai.v32i1.11796
7. Hou, Y., Liu, L., Wei, Q., Xu, X., Chen, C. A Novel DDPG Method with Prioritized Experience Replay. IEEE International Conference on Systems, Man, and Cybernetics, 2017, 316-321. https://doi.org/10.1109/SMC.2017.8122622

8. Huang, G., Zhang, Z., Yan, W., Cui, R., Zhang, S. Robust Differential Graphical Games of Uncertain Multiagent Systems. IEEE Transactions on Circuits and Systems II: Express Briefs, 2024, 71(5), 2664-2668. https://doi.org/10.1109/TCSII.2023.3344535
9. Huang, G., Zhang, Z., Yan, W., Guo, X. Differential Graphical Games of Multiagent Systems with Nonzero Leader's Control Input and External Disturbances. International Journal of Robust and Nonlinear Control, 2024, 34(12), 8144-8162. https://doi.org/10.1002/rnc.7378
10. Iqbal, S., Sha, F. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. Proceedings of the 36th International Conference on Machine Learning Research, 2019, 2961-2970. https://doi.org/10.48550/arXiv.1810.02912
11. Kaan, G., Hakan, G. Generalized Huber Loss for Robust Learning and Its Efficient Minimization for a Robust Statistics. arXiv Preprint, 2021. https://doi.org/10.48550/arXiv.2108.12627
12. Kang, C., Rong, C., Ren, W., Huo, F., Liu, P. Deep Deterministic Policy Gradient Based on Double Network Prioritized Experience Replay. IEEE Access, 2021, 9, 60296-60308. https://doi.org/10.1109/ACCESS.2021.3074535
13. Lin, L. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine Learning, 1992, 8(3), 293-321. https://doi.org/10.1007/BF00992699
14. Liu, R., Zou, J. The Effects of Memory Replay in Reinforcement Learning. Proceedings of the Annual Allerton Conference on Communication, Control, and Computing, 2018, 478-485. https://doi.org/10.1109/ALLERTON.2018.8636075
15. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., Mordatch, I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. Proceedings of the 31th International Conference on Neural Information Processing Systems, 2017, 6379-6390. https://doi.org/10.48550/arXiv.1706.0275
16. Lu, J., Zhao, Y. B., Kang, Y., Wang, Y., Deng, Y. Strategy Generation Based on DDPG With Prioritized Experience Replay for UCAV. International Conference on Advanced Robotics and Mechatronics, 2022, 157-162. https://doi.org/10.1109/ICARM54641.2022.9959220
17. Mnih, V., Kavukcuoglu, K., Silver, D. Human-Level Control Through Deep Reinforcement Learning. Nature, 2015, 518(7540), 529-533. https://doi.org/10.1038/nature14236
18. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. Playing Atari with Deep Reinforcement Learning. arXiv Preprint, 2013. https://doi.org/10.48550/arXiv.1312.5602
19. Moore, A. W., Atkeson, C. G. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time. Machine Learning, 1993, 13(1), 103-130. https://doi.org/10.1007/BF00993104
20. Saglam, B., Mutlu, F. B., Cicek, D. C., Kozat, S. S. Actor Prioritized Experience Replay. Journal of Artificial Intelligence Research, 2024, 78. https://doi.org/10.1613/jair.1.14819
21. Schaul, T., Quan, J., Antonoglou, I., Silver, D. Prioritized Experience Replay. International Conference on Learning Representations, 2016. http://arxiv.org/abs/1511.05952
22. Seijen, H. V., Sutton, R. S. Planning by Prioritized Sweeping With Small Backups. arXiv Preprint, 2013. https://doi.org/10.48550/arXiv.1301.2343
23. Sharma, J., Andersen, P. A., Granmo, O. C., Goodwin, M. Deep Q-Learning With Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2021, 51(12), 7363-7381. https://doi.org/10.1109/TSMC.2020.2967936
24. Shen, K. H., Tsai, P. Y. Memory Reduction Through Experience Classification for Deep Reinforcement Learning with Prioritized Experience Replay. IEEE International Workshop on Signal Processing Systems, 2019, 166-171. https://doi.org/10.1109/SiPS47522.2019.9020610
25. Silver, D., Huang, A., Maddison, C. J. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature, 2016, 529(7587), 484-489. https://doi.org/10.1038/nature16961
26. Sovrano, F., Raymond, A., Prorok, A. Explanation-Aware Experience Replay in Rule-Dense Environments. IEEE Robotics and Automation Letters, 2022, 7(2), 898-905. https://doi.org/10.1109/LRA.2021.3135927
27. Tao, X., Hafid, A. S. Deep Sensing: A Novel Mobile Crowdsensing Framework with Double Deep Q-Network and Prioritized Experience Replay. IEEE Internet of Things Journal, 2020, 7(12), 11547-11558. https://doi.org/10.1109/JIOT.2020.3022611
28. Tesauro, G. Extending Q-Learning to General Adaptive Multi-Agent Systems. Proceedings of the 16th

International Conference on Neural Information Processing Systems, 2004, 871-878.

29. Vaibhav, P., Kailasam, L. A Deep Actor Critic Reinforcement Learning Framework for Learning to Rank. Neurocomputing, 2023, 547, 126314. https://doi.org/10.1016/j.neucom.2023.126314
30. Vanseijen, H., Sutton, R. S. A Deeper Look at Planning as Learning from Replay. Proceedings of the International Conference on Machine Learning, 2015, 2314-2322.
31. Xing, X., Zhou, Z., Li, Y., Xiao, B., Xun, Y. Multi-UAV Adaptive Cooperative Formation Trajectory Planning Based on an Improved MATD3 Algorithm of Deep Reinforcement Learning. IEEE Transactions on Vehicular Technology, 2024, 73(9), 12484-12499. https://doi.org/10.1109/TVT.2024.3389555
32. Yang, R., Wang, D., Qiao, J. Policy Gradient Adaptive Critic Design with Dynamic Prioritized Experience Replay for Wastewater Treatment Process Control. IEEE Transactions on Industrial Informatics, 2022, 18(5), 3150-3158. https://doi.org/10.1109/TII.2021.3106402