Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 7619483, 10 pages
https://doi.org/10.1155/2019/7619483

Research Article

A Dynamic Adjusting Reward Function Method for Deep Reinforcement Learning with Adjustable Parameters

Zijian Hu, Kaifang Wan, Xiaoguang Gao, and Yiwei Zhai

School of Electronic and Information, Northwestern Polytechnical University, Xi'an 710129, China

Correspondence should be addressed to Kaifang Wan; wankaifang@nwpu.edu.cn

Received 28 June 2019; Revised 20 September 2019; Accepted 5 November 2019; Published 23 November 2019

Academic Editor: Jose de Jesus Rubio

In deep reinforcement learning, network convergence speed is often slow and easily converges to local optimal solutions. For an environment with reward saltation, we propose a magnify saltatory reward (MSR) algorithm with variable parameters from the perspective of sample usage. MSR dynamically adjusts the rewards for experience with reward saltation in the experience pool, thereby increasing an agent's utilization of these experiences. We conducted experiments in a simulated obstacle avoidance search environment of an unmanned aerial vehicle and compared the experimental results of deep Q-network (DQN), double DQN, and dueling DQN after adding MSR. The experimental results demonstrate that, after adding MSR, the algorithms exhibit a faster network convergence and can obtain the global optimal solution easily.

1. Introduction

Reinforcement learning (RL), a stimulating area of artificial intelligence, aims to improve the action of an agent based on the reward received from an environment [1]. The agent performs an action to obtain reward from the environment and adjusts its policy based on the reward. Through this continuous interaction with the environment, the agent can learn the optimal policy to obtain the maximum cumulative reward. Currently, RL has achieved notable success in many applications.

RL primarily includes two types of algorithms: one is based on value function estimation and the other is based on policy representation, also called policy gradient method [2]. The algorithm based on value function estimation primarily includes the Q-learning algorithm proposed by Watkins and Dayan in 1992 [3] and the Sarsa algorithm proposed by Rummery and Niranjan in 1994 [4]. The advantage of this type of algorithm is that the process is simple and easily implementable; however, it does not solve the problem of continuous action space well. The REINFORCE algorithm proposed by Sutton et al. [5] and the actor-critic algorithm proposed by Konda and Tsitsiklis [6] are typical algorithms based on policy gradient method. The main idea of these algorithms is to link the parameterization policy with the cumulative reward and continuously optimize the policy to obtain the optimal policy. This type of method can solve the problem of continuous action space, but can easily converge to the local optimal solution [7].

The traditional RL algorithms mentioned earlier are often not effective in large-scale complex environments. In 2013, DeepMind innovatively combined deep learning (DL) with RL to form a new research hotspot in the field of artificial intelligence, namely deep reinforcement learning (DRL). By leveraging the perceived capabilities of DL and the decision-making capabilities of RL, DRL can train agents that are comparable to humans and even beyond human levels. DRL has recently been applied to solve challenging problems, including Atari games [8, 9], man-machine confrontations [10–12], mobile crowdsensing [13, 14], autonomous navigations [15, 16], and robot control [17–19].

In DRL, an agent must interact with the environment to adapt to the environment, thus causing the agent to spend a considerable amount of time to attain the optimal behavior. In this process, the agent learns how to move in the environment based on the rewards. To accelerate this learning process, it is hoped that a reward function can describe the agent's state in a timely and accurate manner. Therefore, the design of a reward function has become a key aspect of RL.

Researchers have proposed different methods to design reward functions to accelerate learning and training. Wei et al. designed a heuristic reward function for RL algorithms [20]. Yan et al. used a hierarchical RL (HRL) algorithm based on a heuristic reward function to solve the problem of huge state space [21]. Maja proposed a methodology for designing reinforcement functions that utilize implicit domain knowledge to accelerate learning [22].

Owing to the complexity of an environment, it is difficult to design a perfect reward function for a specific environment. Mestol et al. proposed an algorithm that is a variation of RL with a reward adapted to the degree of uncertainty of a performed prediction and solved some problems that could not be solved using traditional RL [23]. Wu et al. proposed an adaptive network scaling framework to obtain a suitable scale of rewards during learning for a better performance [24]. In a system based on the principle of indirect reciprocity, Xiao et al. designed a method to update the reward of nodes, which greatly increased the security of the system [25]. We herein focus on the optimization of the reward function based on the existing reward function to achieve better results. By observing the changes in rewards during the RL process, we discovered that rewards often change significantly, especially when an agent succeeds or fails. Unlike the abovementioned studies, we consider the psychological reaction of people when they experience success or failure. Furthermore, we propose a dynamic adaptive parameter optimization algorithm, i.e., magnify saltatory reward (MSR) to apply to the abovementioned phenomenon for improving the performance of algorithms. MSR primarily focuses on samples with saltatory rewards and modifies the rewards in real time during learning according to saltation size. The effectiveness of MSR is tested in various environments, and different deep reinforcement learning algorithms, including DQN, double DQN, and dueling DQN, are examined in simulated obstacle avoidance search environment of an UAV. The experiment results suggest that, after adding MSR, the convergence speed of the networks is improved by 7–27% and the convergence results of the algorithms are improved by 3–10%.

2. Preliminaries

2.1. Markov Decision Process. The Markov decision process (MDP), a model for sequential decisions, is proposed by Bellman in 1957 [26]. Almost all the RL problems can be modeled as an MDP. The MDP is shown as a 5-tuple (S, A, T, R, γ), in which each part is defined as follows [1]: S, the state that an agent can attain in a specific environment; A, the action that an agent must execute to move from one state to another; T, the probability of executing an action a to move an agent from state s to state s'; $R(s, a, s')$, the received reward by an agent after executing action a and transiting from state s to state s'; γ, the discount factor that determines the importance of current and future rewards.

2.2. RL and Q-Learning. The first aim in RL is to transform the problem into an MDP. The next step is to obtain an optimal policy for this problem [27]. Q-learning [3] and Sarsa [4] are the most well-known RL.

We define discrete time steps as follows: $t = 1, 2, 3 \ldots$, where at each time step t, the agent interacts with the environment to obtain the environment state S_t, where $S_t \in S$ and S is the collection of all environmental states. Subsequently, the agent selects an action a_t based on the current state S_t, where $a_t \in a(S_t)$ and $a(S_t)$ is the collection of actions that can be selected at state S_t. Next, the agent performs the action a_t, and the state of the environment reaches S_{t+1}. Simultaneously, the agent will obtain a reward r_t. The above process is looped continuously until the agent reaches the terminal. Suppose the duration of an episode is from time t to T; therefore, the cumulative reward obtained by the agent in this episode is obtained as follows:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{1}$$

where γ is a discount factor and its value range is $[0, 1]$. The role of the discount factor is to weigh the impact of future rewards on the cumulative rewards. The action value function is defined as follows:

$$Q(s, a) = E[R_t \mid s_t = s, a_t = a, \pi]. \tag{2}$$

Equation (2) represents the cumulative reward obtainable by agent if the agent executes action a at state s and always follows the policy π to the end of the episode. The update formula of the action value function is defined as follows:

$$Q(s, a) \longleftarrow Q^{\pi}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]. \tag{3}$$

Equation (3) is used to iterate and finally obtain the optimal action value function of Q-learning.

2.3. Deep Q-Network. To expand the scope of Q-learning, Mnih et al. combined a convolutional neural network with Q-learning and proposed a DQN model [8, 9]. The DQN has two networks with the same hyperparameters. The evaluation network uses $Q(s, a; \theta)$ as the Q function to approximate the action value function while the target network uses $Q(s, a; \theta^{-})$. The parameters of the evaluation network are updated in real time, and its parameters are copied to the target network at every N iteration. When the number of iterations is i, the loss function is as follows:

$$\begin{aligned} L_i(\theta_i) &= E_{(s,a,r,s')}\left[\left(y_i^{\mathrm{DQN}} - Q(s, a; \theta_i)\right)^2\right], \\ y_i^{\mathrm{DQN}} &= r + \gamma \max_{a'} Q(s', a'; \theta^{-}), \end{aligned} \tag{4}$$

where θ_i represents the network parameters in the evaluation network, while θ_i^{-} represents the parameters in the target network. The main features of the DQN are as follows: (1) DQN, an end-to-end training method, not only fully exerts the powerful feature extraction capabilities of deep neural networks but also reflects the ability of RL to adapt to unknown environments [7]. (2) A DQN is implemented by

integrated experience playback technology [28]. Experience playback increases the efficiency of data usage by sampling historical data, while reducing the correlation between data. (3) The target and evaluation networks use the same hyperparameters. The network structure and algorithm can be applied to many different tasks, thus implying the strong versatility of the algorithm.

Q-learning overestimates the action value in large-scale data owing to the estimation error inherent during learning [29]. The double deep Q-network (double DQN) proposed by Van Hasselt et al. [30] applies two Q-learning methods to the DQN. It effectively avoids overestimation and obtains a more stable and effective policy. The Q value of the target network in the double DQN algorithm is shown in the following equation:

$$y_i^{\text{DoubleDQN}} = r + \gamma Q\left(s', \arg\max Q\left(s', a \mid \theta_i\right), \theta_i^-\right). \quad (5)$$

In 2016, Wang et al. proposed a neural network architecture named dueling architecture (dueling DQN) [31] for model-free RL and experimentally proved that the DQN of dueling architecture can yield better policy. The difference in network between DQN and dueling DQN is shown in Figure 1:

As shown in equation (6), the dueling DQN divides the output of the neural network into two parts: the state-value function and the state-dependent action advantage function.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha), \quad (6)$$

where α and β represent the parameters of the state-value function and action advantage function, respectively and θ represents the rest parameters of the network.

3. Algorithm

3.1. Reward Saltation. In RL, the reward obtained by an agent after each action is generally different. We define the phenomenon where the reward of the last action differs vastly from the currently obtained reward as reward saltation. Two types of reward saltation exist: one is positive saltation, that is, the current reward is greater than the reward of the previous step; the other is negative mutation, in which the current reward is smaller than the last reward. Positive saltation often occurs with shortcuts. For example, in a maze game, when a shortcut to an end point is found, the reward of the agent would be changed significantly. Negative saltation generally occurs in unexpected situations, such as when a racing game is taking place, a car accident occurs near the end.

The purpose of artificial intelligence is to make machines more like humans and even beyond humans. If one obtains a new solution to a problem that is more effective and reliable than previous solutions, then one will use this solution for similar problems. If one fails unexpectedly when approaching success, such as accidentally failing while having a high score in a video game, then one will be highly cautious when reaching the high score next time; one might even remain at that state without taking any risks. This is the phenomenon of "deviation from the status quo" in psychology [32]. Besides, the norm theory put forward by Kahneman and Miller in 1986 [33] argues that individual's emotional responses are often more intense because the cause of the decision is nonstandard. In other words, abnormal and unexpected events are more likely to cause an individual's strong emotional response. Therefore, these experiences are important in the growth and progress of a person. As humans do, we hope that the agent can focus on these experiences, that is, the experience of reward saltation. From these experiences, the agent can, like humans, decisively select shortcuts when they are available and be more cautious in performing the next action in potentially dangerous situations.

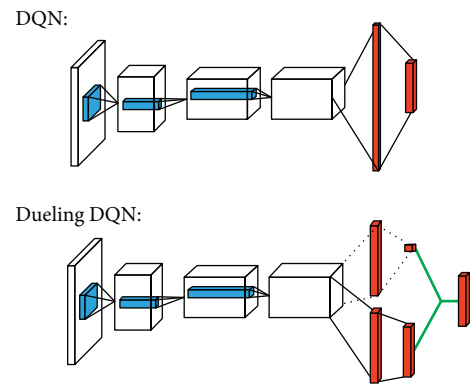

Figure 1: Network structures of DQN and dueling DQN.

Our idea is to magnify the current reward for a positive saltation and narrow down the current reward for a negative saltation, followed by testing the idea in a maze environment. As shown in Figure 2, a maze is a simple RL environment. The red square represents the agent, the black squares represent obstacles, and the green circle represents the terminal. The goal in maze is to have the agent reach the terminal without touching any obstacles. The state is the coordinate information of the red square, and the action space includes four actions of up, down, left, and right. If the agent collides with the wall, it does not change its position; however, if it hits an obstacle, it dies. The reward function is related to the location of the agent, that is, the closer the agent to the end point, the greater is the reward.

We used a DQN as the deep RL algorithm and success rate as the evaluation indicator, that is, we calculated the proportion of successful episodes in all previous episodes per 100 episodes. The experimental results are shown in Figure 3.

The experimental results show that the MSR can effectively accelerate the convergence speed of the deep RL algorithm. This implies that the experience of reward saltation is important for the training of the agent.

3.2. MSR. As mentioned, magnifying the current reward value for a positive saltation and reducing the current reward for a negative saltation affect the network convergence speed of deep RL positively. The next step is to determine when to magnify the reward, how to magnify reward, and the extent

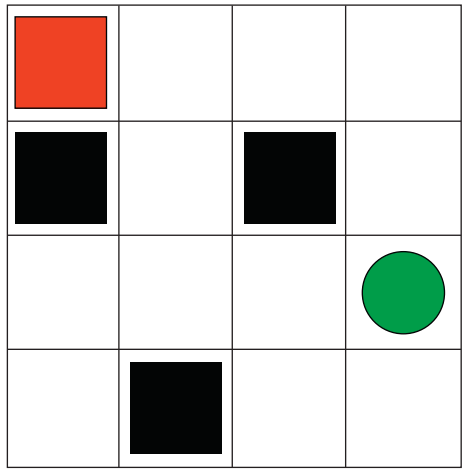

Figure 2: Maze environment.

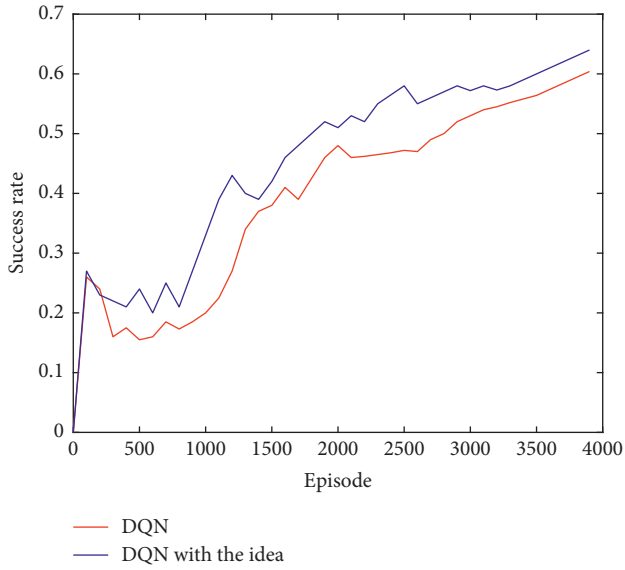

Figure 3: Success rate of DQN and DQN with the idea.

of the magnification. If the reward is magnified significantly in some places, a low degree of exploration of the agent will result in these places, which is not conducive to the network convergence. If magnification is insufficient, the role of the experience of reward saltation will not be fully realized.

First, we define a parameter ρ as follows:

$$\rho = \frac{r_t}{r_{t-1}}, \tag{7}$$

where r_t and r_{t-1} represent the rewards of the current and last action, respectively. We define that when ρ is greater than a certain number η as the reward saltation, the reward must then be magnified. Defining ρ as such causes two problems: (1) In many environments, the reward may be zero, which will cause ρ to be meaningless. (2) The reference point is r_{t-1}. When r_{t-1} is extremely large or extremely small, the value of r_t has little effect on ρ. Therefore, we modify the definition of ρ as

$$\rho = \frac{r'_t - r'_{t-1}}{\min\left(\left|r'_t\right|, \left|r'_{t-1}\right|\right)}, \tag{8}$$

$$r'_t = r_t + \sigma, \tag{9}$$

$$r'_{t-1} = r_{t-1} + \sigma, \tag{10}$$

where σ is a number that approaches zero.

To ensure an appropriate magnification of the current reward, a monotonically increasing bounded function is required. After experimenting, we decide to use the $f(x) = \arctan(x)$ function, whose range is $(-\pi/2, \pi/2)$. Using positive saltation as an example, we wish to magnify r_t when $\rho > \eta$, that is, when $\rho = \eta$, $f(\rho) = 1$, and when $\rho > \eta$, $f(\rho) > 1$. Therefore, the function becomes that in equation (11) and is shown in Figure 4:

$$f(x) = \arctan\left(x * \frac{\pi}{2} * \frac{1}{\eta}\right), \tag{11}$$

where

$$\begin{aligned} x &= \rho + \lambda, \\ \lambda &= \operatorname{sgn}\left(r_t - r_{t-1}\right). \end{aligned} \tag{12}$$

When $x > \eta$, the magnified r_t^* is

$$r_t^* = r'_t + (f(x) - \lambda) * \left|r'_t\right|. \tag{13}$$

Similarly, the r_t^* of a negative saltation can be derived from equation (13).

Overall, using a DQN as an example, we refer to our algorithm as MSR, Algorithm 1. The double DQN with MSR and the dueling DQN with MSR are similar to Algorithm 1.

MSR is a parameter-adjustable optimization algorithm for the reward function. In RL, MSR does not interfere with the interaction between the agent and the environment, but modifies some rewards before storing the experience in the experience pool.

MSR primarily presents the following advantages: (1) It distinguishes the failure and success of different states, thus rendering the reward function more detailed. In general, the reward when an agent fails or succeeds in an environment is fixed, irrespective of its state. MSR associates the failed or successful rewards with the states. As shown in Figure 5(a), the agent will obtain different rewards when hitting the marked obstacles in position ① and position ②, and the reward obtained in position ② is lower. (2) MSR causes the agent to be more cautious when encountering a risk such that failure can be avoided. As shown in Figure 5(b), if the agent fails at position ③ because the position is closely associated to success, saltation in reward will result in the agent obtaining an extremely low reward. The next time the agent arrives at this position, the probability of selecting the action that caused failure will be low. (3) MSR causes the agent to be more decisive when encountering high-yield choices, thus rendering it easier to obtain the optimal solution to the problem. As shown in Figure 5(c), it is assumed that the path originally learned by the agent is that represented by the red line. When the agent accidentally selects

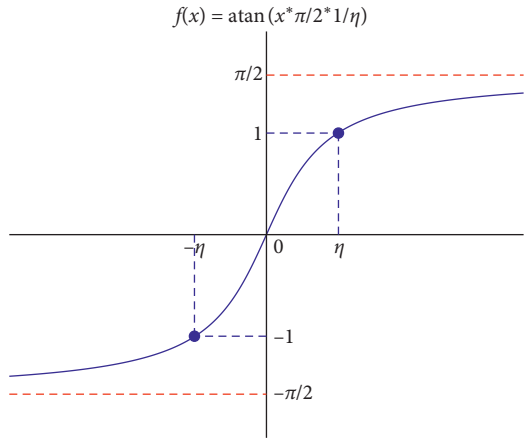

Figure 4: $f(x) = \arctan(x * \pi/2 * 1/\pi)$.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\widehat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Receive initial observation s_1 and initialize η
  For $t = 1$, T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t in an emulator and observe reward r_t
    Set $r'_t = r_t + \sigma$, $\rho = r'_t - r'_{t-1}/\min(|r'_t|, |r'_{t-1}|)$, and $x = \rho + \lambda$
    Update reward $r^*_t = \begin{cases} r'_t + (\arctan(x * \pi/2 * 1/\eta) - \lambda) * |r'_t|, & \text{if } x > \eta \text{ or } x < -\eta, \\ r_t, & \text{otherwise.} \end{cases}$
    Set $s_{t+1} = s_t$, a_t
    Store transition $(s_t, a_t, r^*_t, s_{t+1})$ in D
    Sample random minibatch of transition (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1, \\ r_j + \gamma \max_{a'} \widehat{Q}(s_{j+1}, a'; \theta^{-}), & \text{otherwise.} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\widehat{Q} = Q$
  End For
End For

Algorithm 1: MSR algorithm (with DQN).

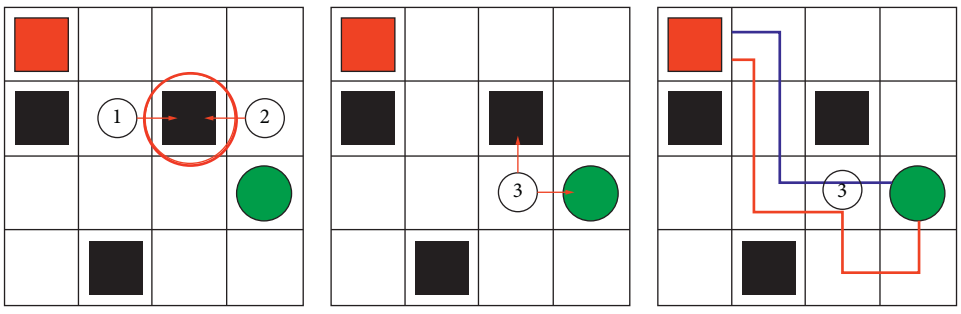

Figure 5: Maze example.

the rightward action at position ③, the saltation of the reward causes the agent to obtain a high reward. The next time the agent arrives at position ③, the probability of selecting to move to the right will be greater, which will make the path resemble that represented by the blue line. (4) Variable parameters allow MSR to be adaptable. In the MSR algorithm, the main adjustable parameter is η, which determines when the reward should be modified. Therefore, selecting a more suitable η for the current environment can yield the optimal solution faster. Figure 6 shows the success rate of different η in the maze environment. As shown, MSR can accelerate the convergence of neural networks in the maze environment, and the best effect is when η is 2.

3.3. Test in Gym Environment. We tested our approach on OpenAI Gym (https://gym.openai.com), which is a toolkit typically used in RL to test algorithms. LunarLander-v2 is an environment of Box2D in Gym. As shown in Figure 7, the agent is a lunar lander with three engines (left, right, and down), and our goal is to make it land safely and smoothly in the designated area. An episode is completed if the lander crashes or comes to rest; thus, it receives additional −100 or +100 points. Each leg ground contact corresponds to +10 points. Firing the main engine gives −0.3 points in each frame. 200 points will be obtained if the problem is solved.

The hyperparameters are set as follows: Each time, 32 experiences are selected from the experience pool; the discount factor is set to 0.99, the learning rate is 0.0001, the memory size is 250000, and the total training round is 5000. The exploration factor is set to a form that decreases as the number of steps increases (from 1 to 0.01). The maximum number of steps per episode is 1000 to prevent an infinite loop.

Approximately seven hours are required for one round of training. After performing many tests, the experimental results are as shown in Figure 8. It is clear that the DQN after adding the MSR algorithm achieves convergence of approximately 3000 rounds, and the original DQN reaches convergence at about 4000 rounds. The accumulated reward per episode after convergence was significantly improved: that of the original DQN was approximately 170 and DQN with MSR was approximately 220.

We performed many experiments to compare the effects of different x on the algorithm for this environment. As shown in Figure 9, different η values have different effects on the algorithm, and the most suitable η value for this environment is approximately 5. In addition, it is clear that the results of MSR are generally better than those of the DQN in terms of convergence rate and convergence results.

4. Experiments

4.1. Experimental Platform. The analysis and experiments presented prove that MSR affects DQN positively. Subsequently, we tested MSR in a more complex and practical environment. We conducted experiments in a two-dimensional unmanned aerial vehicle (UAV) obstacle avoidance search environment. The main task of the environment is to enable the UAV to search the target while avoiding obstacles. The UAV motion model is shown in Figure 10:

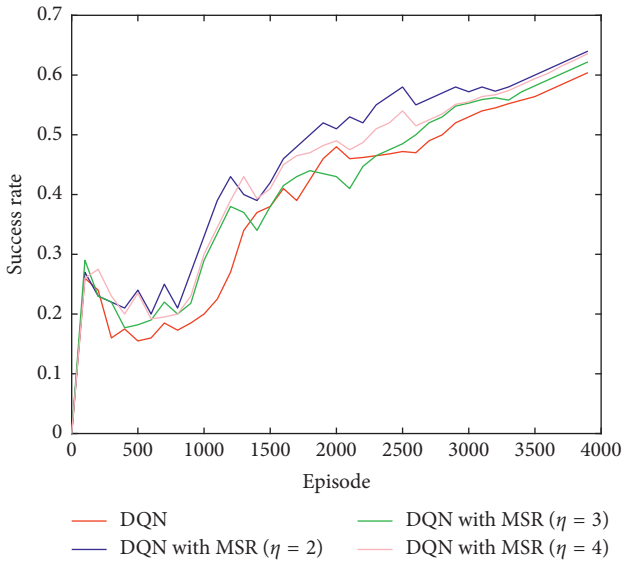

Figure 6: Success rate of DQN with MSR (different η).

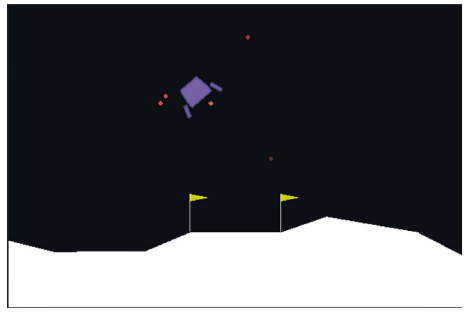

Figure 7: LunarLander-v2.

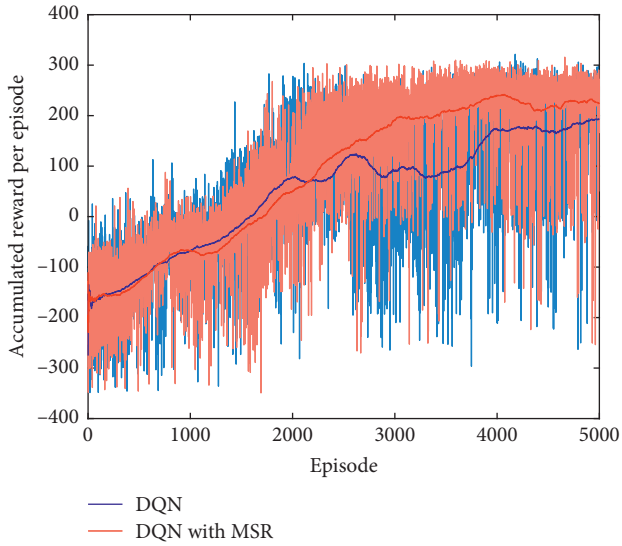

Figure 8: Accumulated reward per episode of DQN and DQN with MSR.

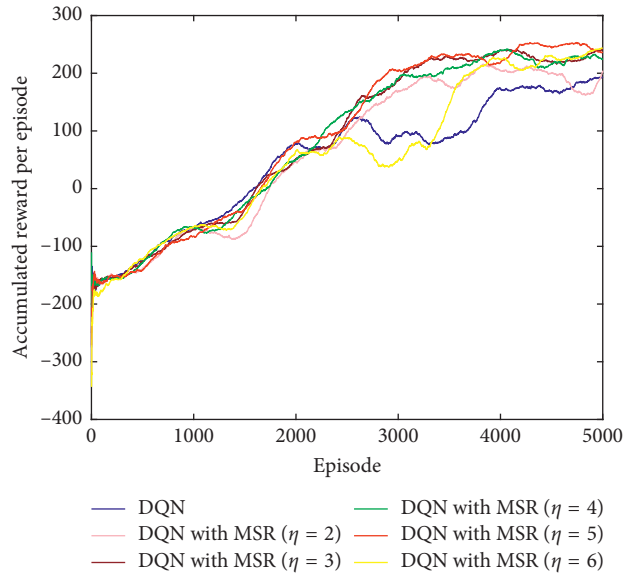

Figure 9: Accumulated reward per episode of DQN (different η).

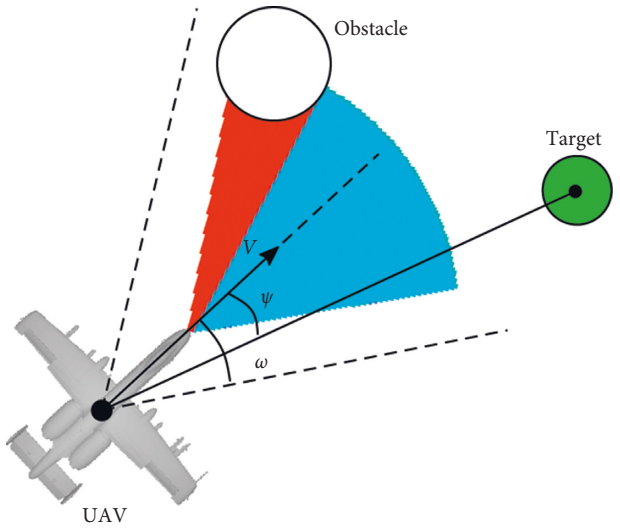

Figure 10: UAV model.

The UAV is equipped with radar for detecting targets and obstacles, and the approximate position of the target has been determined in advance using ground radar and satellite. At the beginning of each episode, the location of the UAV and the target are randomly generated and the number, radius, and location of the obstacles are fixed. The input of the network is the environmental information acquired by the radar and the relative position information of the UAV and the target, where the output of the network is the action of the UAV. Within each motion step, the UAV can choose to maintain the current flight direction, clockwise ω degree or counterclockwise ω degree. The specific simulation environment is shown in Figure 11.

In the experiment, the reward function can be expressed as follows:

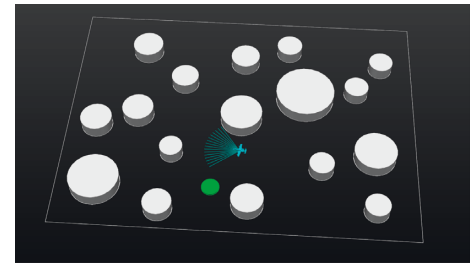

Figure 11: Simulation environment.

$$r(s_t, a_t) = \begin{cases} r_a, & \text{if arrive the target,} \\ r_b, & \text{if collide,} \\ \mu_1 * d + \mu_2 * \psi + \mu_3 * d_d, & \text{every step.} \end{cases} \tag{14}$$

If the UAV finds the target and reaches the target, it will obtain a positive reward r_a. If the UAV hits an obstacle or boundary during the search, it will obtain a negative reward r_b. In other cases, rewards are related to the following factors: (1) d, amount of change in the relative position between the UAV and the target; (2) ψ, angle between the current velocity direction and the actual direction of the target; (3) d_d, distance between the target (or obstacle) and the UAV when the target (or obstacle) is detected; μ_1, μ_2, and μ_3 are three fixed coefficients corresponding to d, ψ, and d_d, respectively.

Our hyperparameters are set as follows: Each time, 32 experiences are selected from the experience pool; the discount factor is set to 0.9, the learning rate is 0.0001, the memory size is 50000, and the total training round is 5000. The exploration factor is set to a form that decreases as the number of steps increases (from 1 to 0.05).

The CPU of our experimental hardware platform is Intel I7-9700K, the GPU is RTX-2070, the memory is 32 G, and the operating system is Ubuntu 16.04.

4.2. Results and Analysis. To prove the universality of MSR, we used the DQN, double DQN, and dueling DQN algorithms for the same simulation environment. The results of the original algorithm and the algorithm after adding MSR ($\eta = 2$) are compared. As UAV and target positions are randomly generated, the number of steps per episode is different; therefore, we used the hit rate and the average reward of each episode as the evaluation indexes. The hit rate of a current episode is defined as the success rate of the previous 100 episodes. The episode average reward is calculated by dividing the episode accumulated reward by the number of steps in that episode. The experimental results are shown in Figure 12:

To more intuitively demonstrate the advantages of MSR, we define E_c as the episode number of the convergence, and we use the first time when the hit rate reaches 80% as the criterion for convergence. Additionally, we define R_a as the average reward after complete convergence and we consider 4000–5000 episodes as the fully convergent episodes. These two parameters are used to quantify the promotion of different RL algorithms after adding MSR (Table 1).

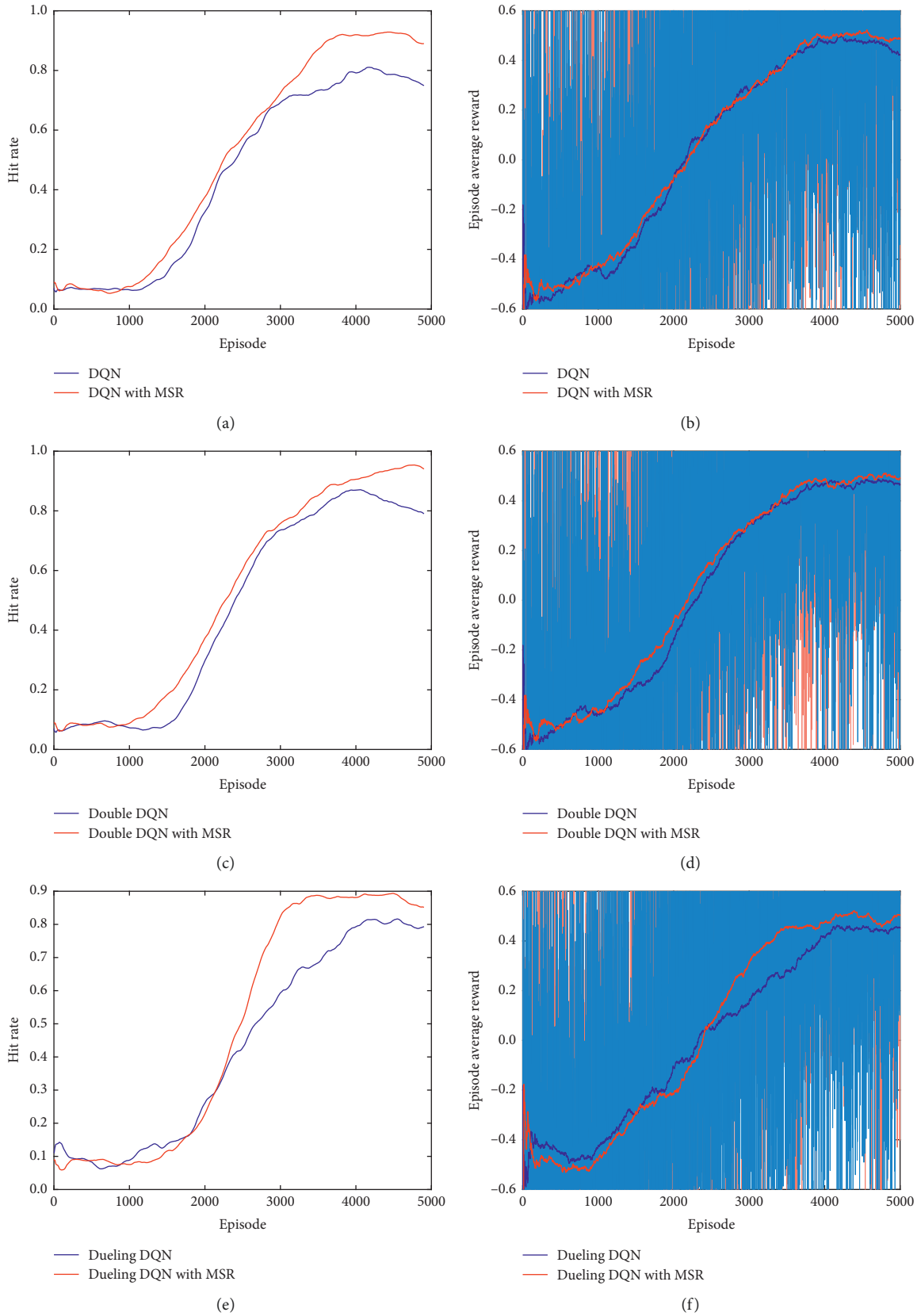

Figure 12: Experimental results. (a) Hit rate of DQN. (b) Episode average reward of DQN. (c) Hit rate of double DQN. (d) Episode average reward of double DQN. (e) Hit rate of dueling DQN. (f) Episode average reward of dueling DQN.

Table 1: Results of different algorithms with MSR.

	Original (E_c)	MSR (E_c)	Promotion (E_c)	Original (R_a)	MSR (R_a)	Promotion (R_a)
DQN	4085	**3288**	**19.51%**	0.4700	**0.4977**	**5.89%**
Double DQN	3500	**3257**	**6.94%**	0.4718	**0.4863**	**3.07%**
Dueling DQN	4065	**2954**	**27.33%**	0.4468	**0.4921**	**10.14%**

The experimental results show that, for different RL algorithms and after adding MSR, both the convergence speed and the training results improved. Among these three algorithms, the effect of MSR on the dueling DQN is the most obvious.

5. Conclusions

The slow convergence of the network and the tendency to fall into the local optimal solution are the areas of improvement in DRL. The MSR algorithm proposed herein adjusts the rewards of the experience with reward saltation in the experience pool, which accelerates the convergence speed of the network and allows the network to more easily converge to the global optimal solution. In addition, the adjustable parameters in the MSR rendered it adaptability to various environments that contain reward saltation, and the experimental results proved that MSR exhibited significant improvement in training after combining with different DRL algorithms.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are grateful to Professor Gao and Dr. Wan for the discussions. This study was supported by the National Natural Science Foundation of China (grant no. 61573285) and the Science and Technology on Avionics Integration Laboratory and Aeronautical Science Foundation of China (ASFC) (grant no. 20175553027).

References

[1] S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA, 1998.

[2] Q. Liu, J.-W. Zhai, and Z.-Z. Zhang, "A survey on deep reinforcement learning," *Chinese Journal of Computers*, vol. 41, no. 1, pp. 1–27, 2018.

[3] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[4] G. Rummery and M. Niranjan, "On-line Q learning using connectionist systems," Technical Report CUDE/F-INFENG/TR 166, Cambridge University Engineering Department, Cambridge, UK, 1994.

[5] R. S. Sutton, D. A. McAllester, S. P. Singh et al., "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, pp. 1057–1063, MIT Press, Cambridge, MA, USA, 2000.

[6] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, pp. 1008–1014, MIT Press, Cambridge, MA, USA, 2000.

[7] Y.-N. Zhao, P. Liu, W. Zhao, and X.-L. Tang, "Twice sampling method in deep Q-network," *Acta Automatica Sinica*, vol. 45, no. 10, pp. 1870–1882, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, https://arxiv.org/abs/1312.5602.

[9] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[11] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," *Control Theory & Applications*, vol. 34, no. 12, pp. 1529–1546, 2017.

[13] L. Xiao, Y. Li, G. Han, H. Dai, and H. V. Poor, "A secure mobile crowdsensing game with deep reinforcement learning," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 35–47, 2018.

[14] Y. Zhan, Y. Xia, J. Zhang, L. Ting, and W. Yu, "Crowdsensing game with demand uncertainties: a deep reinforcement learning approach," 2018, https://arxiv.org/abs/1901.00733.

[15] Y. Zhu, R. Mottaghi, E. Kolve et al., "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3357–3364, Singapore, May 2017.

[16] K. Mo, H. Li, Z. Lin, and J.-Y. Lee, "The adobeindooenav dataset: towards deep reinforcement learning based real-world indoor robot visual navigation," 2018, https://arxiv.org/abs/1802.08824.

[17] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: deep inverse optimal control via policy optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 49–58, New York, NY, USA, June 2016.

[18] F. Zhang, L. Jürgen, M. Michael, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," 2015, https://arxiv.org/abs/1511.03791.

[19] S. Gu, H. Ethan, L. Timothy et al., "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016, https://arxiv.org/abs/1610.00633.

[20] Y. Wei, M. Zhao, F. Zhang, and Y. Hu, "Research of a heuristic reward function for reinforcement learning algorithms," in *Proceedings of the Intelligent Control and Automation*, pp. 2676–2680, Hangzhou, China, June 2004.

[21] Q. Yan, Q. Liu, and D. Hu, "A hierarchical reinforcement learning algorithm based on heuristic reward function," in

Proceedings of the IEEE International Conference on Advanced Computer Control, Shenyang, China, March 2010.

[22] J. M. Maja, "Reward functions for accelerated learning," in *Machine Learning Proceedings 1994*, pp. 181–189, Elsevier, Amsterdam, Netherlands, 1994.

[23] B. Mesot, E. Sanchez, C. A. Pena, and A. Perez-Uribe, *SOS++: finding smart behaviors using leaning and evolution*, MIT Press, Cambridge, UK, 2002.

[24] Y.-H. Wu, F.-Y. Sun, Y.-Y. Chang, and S.-D. Lin, "ANS: adaptive network scaling for deep rectifier reinforcement learning models," 2018, https://arxiv.org/abs/1809.02112.

[25] L. Xiao, Y. Chen, W. S. Lin, and K. J. R. Liu, "Indirect reciprocity security game for large-scale wireless networks," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 4, pp. 1368–1380, 2012.

[26] R. Bellman, "A Markovian decision process," *Indiana University Mathematics Journal*, vol. 6, no. 4, pp. 679–684, 1957.

[27] M. Marashi, A. Khalilian, and M. E. Shiri, "Automatic reward shaping in reinforcement learning using graph analysis," in *Proceedings of the IEEE International Conference on Computer and Knowledge Engineering*, Mashhad, Iran, 2012.

[28] L. J. Lin, *Reinforcement Learning for Robots Using Neural Network*, Carnegie Mellon University, Pittsburgh, PA, USA, 1993.

[29] D.-b. Zhao, K. Shao, Y.-h. Zhu et al., "Review of deep reinforcement learning and discussions on the development of computer Go," *Control Theory & Applications*, vol. 33, no. 6, pp. 701–717, 2016.

[30] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2094–2100, Phoenix, AZ, USA, February 2016.

[31] Z. Wang, N. Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, New York, NY, USA, June 2016.

[32] W. Samuelson and R. Zeckhauser, "Status quo bias in decision making," *Journal of Risk and Uncertainty*, vol. 1, no. 1, pp. 7–59, 1988.

[33] D. Kahneman and D. T. Miller, "Norm theory: comparing reality to its alternatives," *Psychological Review*, vol. 93, no. 2, pp. 136–153, 1986.