Advance Access publication on 15 August 2019
doi: 10.1093/comjnl/bxz066

Deep Reinforcement Learning with Adaptive Update Target Combination

Z. Xu, L. Cao* and X. Chen

Institute of Command and Control Engineering, Army Engineering University, Nanjing, 210007, China
**Corresponding Author: caolei.nj@foxmail.com*

Simple and efficient exploration remains a core challenge in deep reinforcement learning. While many exploration methods can be applied to high-dimensional tasks, these methods manually adjust exploration parameters according to domain knowledge. This paper proposes a novel method that can automatically balance exploration and exploitation, as well as combine on-policy and off-policy update targets through a dynamic weighted way based on value difference. The proposed method does not directly affect the probability of a selected action but utilizes the value difference produced during the learning process to adjust update target for guiding the direction of agent's learning. We demonstrate the performance of the proposed method on CartPole-v1, MountainCar-v0, and LunarLander-v2 classic control tasks from the OpenAI Gym. Empirical evaluation results show that by integrating on-policy and off-policy update targets dynamically, this method exhibits superior performance and stability than does the exclusive use of the update target.

Keywords: Exploration and exploitation; Q-learning; Sarsa; update target; value difference

Received 28 September 2018; Revised 2 April 2019; Editorial Decision 3 April 2019
Handling editor: Jin-Hee Cho

1. INTRODUCTION

The goal of reinforcement learning agent is to maximize accumulated rewards from the environment through trial and error. It is necessary to explore the environment sufficiently and find better action for an agent. One of the most critical problems in reinforcement learning is balancing the exploration and exploitation. Exploration in reinforcement means that the agent tries to select a new action for potential high rewards in the long term, while exploitation means that the agent utilizes acquired knowledge to choose the best action for maximizing short-term rewards. When to explore or exploit is a core challenge for designing a powerful RL algorithm.

Most of the classic reinforcement learning algorithms use simple exploitation strategies such as uniform sampling [1], independent identically distributed/correlated Gaussian noise [2], and randomized least-squares value iteration (RLSVI) [3]. These strategies will be costly when the state space size of tasks is very large. Recently developed exploration strategies for deep reinforcement learning algorithms have led to the significantly improved performance on environments with high-dimensional state spaces. Osband *et al.* [4] developed bootstrapped DQN, which combined deep exploration with deep neural networks for efficient exploration and substantially improved learning speed and cumulative performance across most games in the Arcade Learning Environment. Houthooft *et al.* [5] proposed a novel method called variational information maximizing exploration (VIME), which encouraged the agent to explore by acquiring information about environment dynamics, and the proposed method performed well on various robotic locomotion problems with sparse rewards. Celiberto *et al.* [6] proposed a heuristic exploration function through case-based reasoning, and Bianchi *et al.* [7] converted existing similar cases or trajectories to heuristics. Intrinsic motivation methods [8] used pseudo-counts to achieve state-of-the-art performance on *Montezuma's Revenge*, an extremely challenging Atari 2600 game. Randomized least-squares value iteration was designed to explore and generalize the state of the environment efficiently via linearly parameterized value functions, and it was proved to be effective in several environments, but this method needs to be extended to nonlinear contexts. Florensa *et al.* [9] utilized stochastic neural networks (SNNs) combined with an intrinsic reward, which was designed according to the domain knowledge about the downstream tasks. Achiam and Sastry [10] assumed to design intrinsic rewards in order to approximate the true KL divergence, and the experiments showed that the proposed method enables the agent to succeed in continuous action tasks. Those exploration strategies all

have a high requirement for the fine-tuning exploration parameters or have to acquire the domain knowledge about tasks in advance. Essentially, the training process of those methods requires the participation of human knowledge.

In this paper, the authors consider reconstructing the update target of reinforcement learning to guide the direction of exploration automatically. There are many variants of the update target of reinforcement learning. Van Hasselt *et al.* [11] proposed a Double DQN (DDQN) architecture, which makes use of double estimators for update target, and the results show that DDQN finds a better policy and has reduced the overestimations. Anschel *et al.* [12] presented the averaged target DQN (ADQN), which uses a weighted average over past learned networks to reduce generalization noise variance, which leads to reduced overestimations, more stable learning process, and improved performance. Recently, Hausknechtand Stone [13] mixed Monte Carlo method with Q-learning method as a novel method, and they proved the effectiveness of mixed target in discrete and continuous tasks. De Asis *et al.* [14] introduced a new parameter to update target, so as to allow the degree of sampling performed by the algorithm at each step, which results in even greater performance.

This paper focuses on how to utilize the agent's learning process to automatically balance the exploration and exploitation with little human knowledge. Learning from predecessors' work, the authors present an efficient algorithm called "Value-difference Based Deep Sarsa and Q Networks (VBDSQN)", which does not directly affect the probability of a selected action but takes advantage of value difference to adapt update target, so as to balance exploration and exploitation. The proposed algorithm has been evaluated on several classic control problems from the OpenAI Gym, and the empirical results show that the VBDSQN algorithm not only has a better performance than Deep Q Networks but also maintains a more stable learning curve than that of the Deep Sarsa Networks algorithm.

The organization of this paper is stated as follows. Section 2 describes the off-policy Q-learning, the on-policy Sarsa algorithm, and a number of deep reinforcement learning, which will be utilized in the experiments. Then, the framework of the proposed Value-difference Based Deep Sarsa and Q Networks is explained in detail. In Section 3, the results of experiments demonstrate that the presented method is efficient and useful. In Section 4, the experimental results are discussed. Finally, the conclusions of this paper are drawn in Section 5.

2. METHODOLOGY

The model of standard reinforcement learning (RL) is shown in Fig. 1. The agent and environment interact continually, and the agent selects an action a in a state s under the policy π. The policy π in reinforcement learning denotes the mapping of environmental states to agent actions. The environment presents a new state s' due to the action a; meanwhile, the environment also offers a reward r; finally, the agent modifies the policy according to the experience. Under a given policy π and time t, the Q value of a certain action a_t in a state s_t can be calculated as

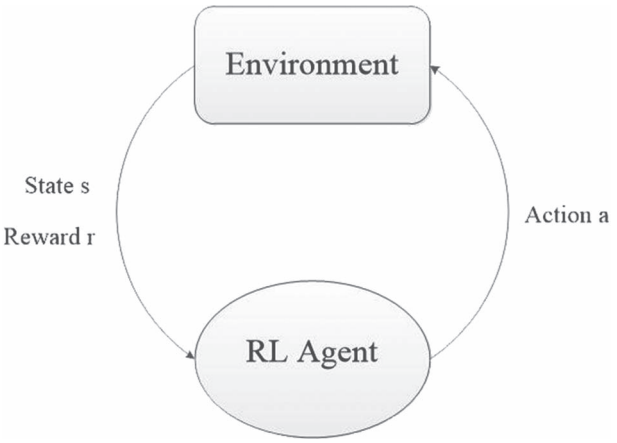

FIGURE 1. The model of standard reinforcement learning.

$$Q_\pi(s_t, a_t) \equiv E[R_1 + \gamma R_2 + \cdots | S_0 = s_t, A_0 = a_t, \pi] \quad (1)$$

where $\gamma \in [0, 1]$ is a discount factor that weighs the importance of immediate and later rewards, R_t refers to the reward given by the environment at the time t, and S_0 and A_0 denote the initial state and action, respectively. The authors assume the optimal value is $Q_*(s, a) = \max_\pi Q_\pi(s, a)$; by selecting the highest-valued action in every state, the agent will easily obtain an optimal policy.

Q-learning is one of the well-known reinforcement learning algorithms, which was proposed by Watkins and Dayan [15]. The simplest form of Q-learning, which is also known as tabular one-step Q-learning, can be defined by the update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \quad (2)$$

where s_t represents the current state, s_{t+1} is the next state, r_t is an immediate reward, α is a learning rate, γ is a discount factor, and $Q(s_t, a_t)$ is an action-value function, which estimates the action value after applying action a_t to state s_t.

Another classic reinforcement learning algorithm is Sarsa, which was proposed by Rummery and Niranjan [16]. The Sarsa algorithm estimates the action-value function $Q(s_t, a_t)$ by applying the following updated formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (3)$$

where s_t represents the current state, s_{t+1} is the next state, r_t is an immediate reward, α is a learning rate, and γ is a

discount factor. This update is done after every transition from a nonterminal state $s_t Q\left(s_{t+1}, a_{t+1}\right)$ is defined as zero if s_{t+1} is terminal.

2.1. Deep reinforcement learning

The sign that the deep reinforcement learning methods began to be widely studied is the proposal of DQN. The DQN algorithm proposed by Mnih *et al.* [17] introduces two critical technologies to improve the learning performance, and one is the target network, and its parameters θ^- will be copied from online networks, $\theta_t^- = \theta_t$, only every τ steps. Then

$$Y_t^{\text{DQN}} \equiv r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a; \theta_t^-\right) \tag{4}$$

where Y_t^{DQN} is the update target of DQN, r_{t+1} is a reward, γ is a discount factor, s_{t+1} is the next state, a is the action, and θ_t^- represents the parameters of the target network at time t.

The other one is experience replay. The agents' experience is stored in the form of $\{s_t, a_t, s_{t+1}, r_t\}$ and sampled uniformly to update the network. These two technologies dramatically improve the performance of the DQN algorithm.

Since the publication of the DQN work, many variants and modification of the standard algorithm have been developed. Schaul *et al.* [18] developed a framework for prioritizing experience to replay important transitions more frequently and therefore learned more efficiently. Wang *et al.* [19] presented a novel neural network architecture called dueling network, which decouples value and advantage in DQN, while sharing a common feature learning module, and leads to dramatic improvements over existing reinforcement learning algorithms. Mnih *et al.* [20] implemented a new variant of the DQN, which uses asynchronous gradient descent for the optimization of deep neural network controllers and shows impressive performance on original DQN. Besides, DSN (Deep Sarsa Networks) [21,22] combines on-policy reinforcement learning algorithm with the deep neural network and is proved to have better performance in some aspects than DQN.

2.2. Mixing updates targets based on value difference

Several simple exploration strategies such as uniform sampling and randomized least-squares are unable to resolve tasks with continuous state or action space. Despite that the VIME exploration strategy has good versatility under dynamic tasks, it is inefficient due to requiring massive amount of environmental information in advance for training. Intrinsic motivation exploration strategies show improvements on *Montezuma's Revenge*, an extremely challenging Atari 2600 game, but the pseudo-counts method needs to manually mark the tasks in the image beforehand and requires accurate prior knowledge. Although a heuristic exploration function through case-based reasoning might improve the efficiency of exploration, it cannot solve complex dynamic tasks. The exploration strategies based on intrinsic rewards improve sample efficiency, but the design of intrinsic rewards is completely based on the knowledge of the human domain. Those strategies all directly adapt exploration strategies manually according to domain knowledge or environmental information without considering the characteristics of the agent's learning process. They have a high requirement for the fine-tuning exploration parameters or have to acquire the environmental information about tasks in advance.

Unlike previous exploration methods, in this paper, the authors propose an efficient and versatile exploration strategy that has no defect in the previous proposed method. The authors combine off-policy Q-learning target and on-policy Sarsa target to take advantage of their own merits. Mixing is accomplished by using a weight coefficient β in [0, 1]. The overall mixed update target is expressed as follows:

$$y = (1 - \beta)\, y_{\text{Q-learning}} + \beta y_{\text{Sarsa}} \tag{5}$$

where y_{Sarsa} represents the on-policy Sarsa target and $y_{\text{Q-learning}}$ represents off-policy Q-learning target. The parameter β is dynamic, which depends on the magnitude of the change in Q value produced during the learning process. The weight coefficient β can be defined as

$$\beta = \left| \frac{e^{\frac{Q_t(s_t,a_t)}{\tau}} - e^{\frac{Q_{t+1}(s_{t+1},a_{t+1})}{\tau}}}{e^{\frac{Q_t(s_t,a_t)}{\tau}} + e^{\frac{Q_{t+1}(s_{t+1},a_{t+1})}{\tau}}} \right| \tag{6}$$

where $Q_t(s_t, a_t)$ is the state action value at time t, $Q_{t+1}(s_{t+1}, a_{t+1})$ is the state action value at time $t + 1$, and τ is a positive constant called inverse sensitivity.

In the process of interaction between the agent and environment, the desired behavior of the agent is to explore more when the knowledge about the environment is uncertain, which is recognized as significant changes in the value function. The exploration efforts should be reduced as the agent's knowledge becomes certain about the environment, which can be recognized as very small or no changes in the value function. In other words, during the learning process, high Q value difference means that the agent should increase exploration, and low Q value difference means the agent should increase exploitation.

Meanwhile, off-policy Q-learning target searches the best possible path by utilizing learned knowledge. The high weights for off-policy Q-learning target are required to help the agent increase exploitation. Sarsa learning target uses the actual Q value for iteration, which strictly updates the value function based on the experience gained by executing certain strategy. Therefore, the high-weighted on-policy Sarsa learning target is required to help the agent increase exploration. The authors related the value difference with the weight coefficient β to maintain the balance between exploration and exploitation, so as to avoid the inaccuracy of directly adjusting the probability of action selection.

TABLE 1. VBDSQN algorithm.

Algorithm 1: Value-difference Based Update Target Deep Sarsa and Q Networks

1. Initialize reply memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For *episode* = 1,...,1 M **do**

4. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

   For $t = 1, \ldots, T$ **do**

5. With probability ε select a random action a_t

   Otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$

6. Execute action a_t in the emulator and observe reward r_t and image x_{t+1}
7. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
8. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
9. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
10. $$y_{Q-learning} = \begin{cases} r_j & \text{if episode stop at step j + 1} \\ r_j + \gamma \hat{Q}\left(\phi_{j+1}, \arg\max\limits_a Q\left(\phi_{j+1}, a; \theta_t\right); \theta_t^{-}\right) & \text{otherwise} \end{cases}$$
11. With probability ε select a random action a_{j+1}

    Otherwise select $a_{j+1} = \arg\max_a Q(\phi_{j+1}, a; \theta)$

12. $$y_{Sarsa} = \begin{cases} r_j & \text{if episode stop at step j + 1} \\ r_j + \gamma \hat{Q}\left(\phi_{j+1}, a_{j+1}, \theta_t^{-}\right) & \text{otherwise} \end{cases}$$
13. $$y_j = \left(1 - \beta\right) y_{Q-learning} + \beta y_{Sarsa}$$
14. $$\beta = \left| \frac{e^{\frac{Q_j\left(\phi_j, a_j\right)}{\tau}} - e^{\frac{Q_{j+1}\left(\phi_{j+1}, a_{j+1}\right)}{\tau}}}{e^{\frac{Q_j\left(\phi_j, a_j\right)}{\tau}} + e^{\frac{Q_{j+1}\left(\phi_{j+1}, a_{j+1}\right)}{\tau}}} \right|$$
15. Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
16. Every C steps reset, $\hat{Q} = Q$

   End For

End For

The full algorithm for training is shown in Table 1. The VBDSQN algorithm improves the standard deep reinforcement learning algorithms in three ways. The first two ways are existing techniques, and the last one is the technology proposed in this paper.

First, VBDSQN uses two technologies known as experience replay and the target network. Experience replay technique stores the tuples of experience at each timestep in the form of $\{s_t, a_t, s_{t+1}, r_t\}$ and in a data set D, pooled over many episodes into a replay memory. By utilizing experience replay, the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters. Besides, the authors use a separate network as the target network for generating the target in the Q-learning update. In every C step, the weights of the online network are copied to the weights of the target network. This modification helps make the neural network more stable.

Second, VBDSQN utilizes the double estimator approach in Double Q-learning [23], which determines the value of the next state, so as to reduce overestimations [24]. Although not fully decoupled, the target network architecture provides a natural candidate for the second value function, without introducing additional networks. VBDSQN evaluates the greedy policy according to the online network, but using the target network to estimate its value.

Finally, a hybrid update target based on value difference was introduced, which combined on-policy and off-policy update target through a dynamic weighted way. If the value difference is high, high weight for on-policy update target will be used to help exploration. If the value difference is low, high weight for off-policy update target will be used to help exploitation.

The agent takes advantage of the value difference produced during the learning process to balance exploration and exploitation. By utilizing the dynamic and hybrid update target, the VBDSQN algorithm is more flexible to adjust the agent's learning direction and improves the learning performance.

Compared to the basic deep reinforcement learning algorithms, the proposed method only introduces an additional parameter β in step 14 of Table 1, the increased time complexity is $o(n^2)$, and the increased space complexity is $o(1)$, which in fact has little negative effect on the efficiency of the basic reinforcement learning algorithms.

3. RESULTS

The VBDSQN algorithm was tested on OpenAI Gym [25] with high-dimensional state space and discrete action space, combined with sparse rewards. OpenAI Gym contains a series of benchmark problems that expose a common interface for testing reinforcement learning algorithms. The proposed algorithm was compared with DQN and DSN algorithms on several classic control tasks. The following parameters were used throughout the experiment. Three algorithms all employed four-layer networks with 100 nodes in each hidden layer, where Rectified Linear Unit (ReLU) was selected as the activation function. The number of input neurons is the state dimension of the task, while the number of output neurons is the action dimension. With the momentum of 0.95 and a network learning rate of 10^{-4}, networks were trained by using the adaptive moment estimation (ADAM) optimizer [26]. Gradients were clipped at 10, scaled as they approached parameter bounds [27]. MSE (mean-square-error) function was selected as the error function of the output layer. In the ε-greedy method, the authors set initial $\varepsilon_{\text{ini}} = 0.5$, and each step is decremented by $\Delta\varepsilon$ until it is equal to ε_{fin}, where $\Delta\varepsilon = \varepsilon_{\text{ini}} - \frac{\varepsilon_{\text{ini}} - \varepsilon_{\text{fin}}}{N}$, N refers to the total training steps, and $\varepsilon_{\text{fin}} = 0.01$. The discount factor $\gamma = 0.99$, and the replay memory capacity was 10,000. Target networks update frequency $C = 300$. For VBDSQN algorithm, the inverse sensitivity $\tau = 5$. In this experiment, the implementation was achieved by using the TensorFlow library [28].

First, the authors evaluated the performance of these three algorithms on the CartPole-v1 task, which is the most commonly used control problem for RL algorithms. Subsequently, the authors demonstrated and compared the performances of algorithms through MountainCar-v0 problem, where a locally optimal solution exists. Finally, the authors also tested the proposed algorithm on a Box 2D game, namely, LunarLander-v2, which is a more complex task than the above classic control problems. These three control tasks have been widely analyzed in reinforcement learning and control literature.

3.1. CartPole-v1

As shown in Fig. 2, CartPole control is a classic control task in reinforcement learning research, where there is an inverted pendulum mounted on a pivot point on a cart. The goal of the control system is to keep the pendulum upright by applying horizontal forces to the cart. The observation consists of four elements: the cart position, the velocity of the cart, the pole angle, and the velocity of the pole angle. The action is a horizontal force of $+1$ or -1 applied to the cart. The reward $r(s, a) = +1$ when the inverted pendulum remains upright for every timestep. The authors set a threshold $\Delta = 1000$ for CartPole-v1, which means that if the pendulum keeps up for 1000 continuous timesteps, the task will be automatically terminated.

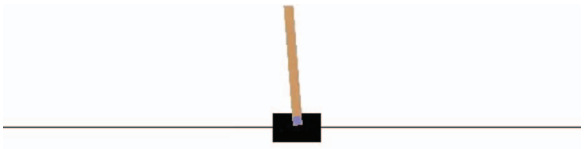

FIGURE 2. CartPole-v1 task.

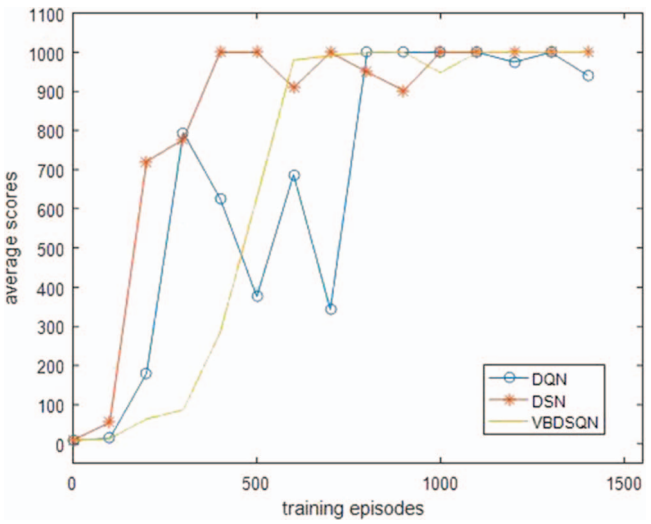

FIGURE 3. The score of CartPole-v1 task.

CartPole-v1 task is a common control task; thus, the authors independently carried out each algorithm 10 times. The learned policy will be tested 10 episodes by every 100 training episodes to calculate the average scores of multiple runs. The score refers to the cumulative reward of agents in each episode.

Figure 3 shows the average score of three algorithms on CartPole-v1 task. We can observe that DSN has reached 1000 points with the fastest learning speed in the first 400 episodes of learning, but the stability of DSN is not as good as VBDSQN from 500 episodes to the end. Due to the simplicity of this task, the advantage of VBDSQN is mainly reflected in the stability. Besides, VBDSQN also shows greater performance than DQN and DSN. The average score and standard deviation of three algorithms are discussed in Section 4.

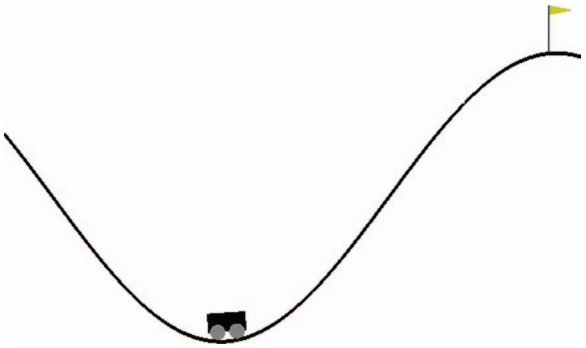

FIGURE 4. MountainCar-v0 task.

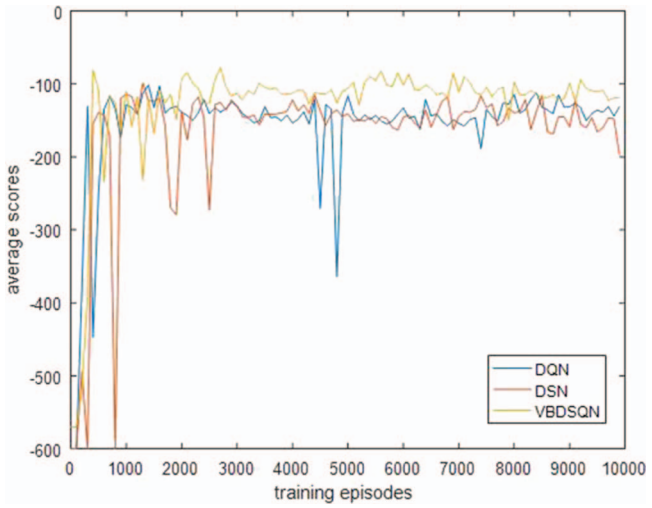

FIGURE 5. The score of MountainCar-v0 task.

3.2. MountainCar-v0

As shown in Fig. 4, in MountainCar-v0 task, the tangential forces are applied to force the car to reach the yellow flag point on the right. Because the maximal tangential force is limited, the car has to alternately drive up along the two slopes of the valley in order to build up enough inertia to overcome gravity. The observation of MountainCar-v0 is given by the horizontal position and the horizontal velocity of the car. The reward $r(s, a) = -1$ is provided for every timestep.

The authors independently carried out each algorithm 10 times. The learned policy will be tested 20 episodes by every 100 training episodes to calculate the average scores of multiple runs. The score refers to the cumulative reward of agents in each episode.

Figure 5 presents the average score of three algorithms on MountainCar-v0 task, compared to DQN; DSN and VBDSQN show more stable performance from 3000 episodes to the end, though DQN has a higher score than DSN. VBDSQN shows the most excellent performance over DQN and DSN on MountainCar-v0 task from 3000 episodes to the end. The specific performance of the three algorithms will be discussed in the following section.

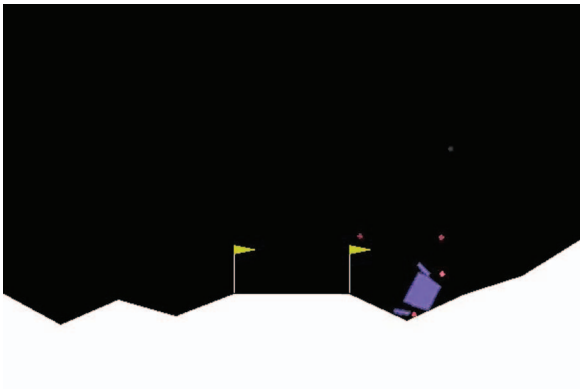

FIGURE 6. LunarLander-v2 task.

3.3. LunarLander-v2

Lunar Lander is an arcade game released by Atari, which uses a vector monitor to display vector graphics [29]. As shown in Fig. 6, in the LunarLander-v2 task, four different discrete actions—do nothing, fire left orientation engine, fire main engine, and fire right orientation engine—are provided during the landing process to help the lander arrive on the landing pad. Fuel is infinite. Thus, the lander can learn to fly and then land on its first attempt. The reward for moving from the top of the screen to the landing pad is $+200$ points. If the lander is away from the landing pad, the reward is 0. The episode terminates if the lander crashes or rests, receiving an extra -100 or $+100$ points. The ground contact for each leg is $+10$ points. Firing main engine will be given -0.3 points every timestep [30].

Due to the complexity of the task, the authors carried out each algorithm 20 times. The learned policy will be tested 50 episodes by every 100 training episodes to calculate the average scores of multiple runs. The score refers to the cumulative reward of agents in each episode.

Figure 7 presents the average score of the three algorithms on LunarLander-v2 task; due to the complexity of the task, in the initial stage of the learning process, all algorithms' performance is unstable. As the knowledge about the environment increases, we can see that VBDSQN gradually shows more excellent performance than do DQN and DSN from 6000 episodes to the end. The average score and standard deviation of three algorithms in LunarLander-v2 task will be detailed in Section 4.

3.4. Sensitivity to hyper-parameters

Figures 8, 9, and 10 show the different values of the parameter inverse sensitivity τ in different tasks. The authors have

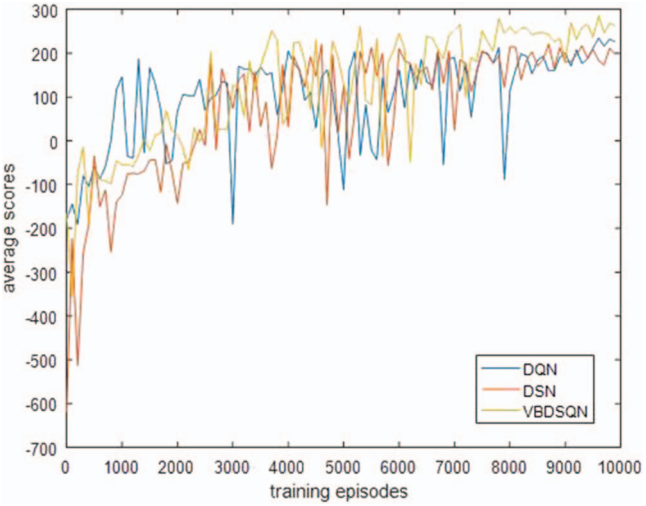

FIGURE 7. The score of LunarLander-v2 task.

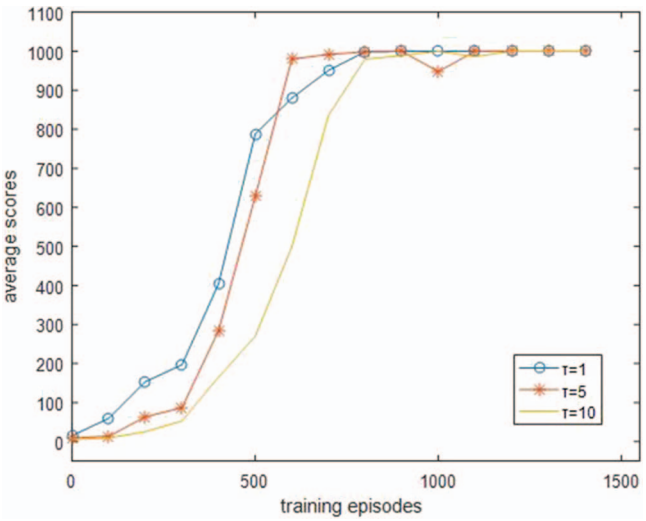

FIGURE 8. The different values of the parameter inverse sensitivity in CartPole-v1 task.

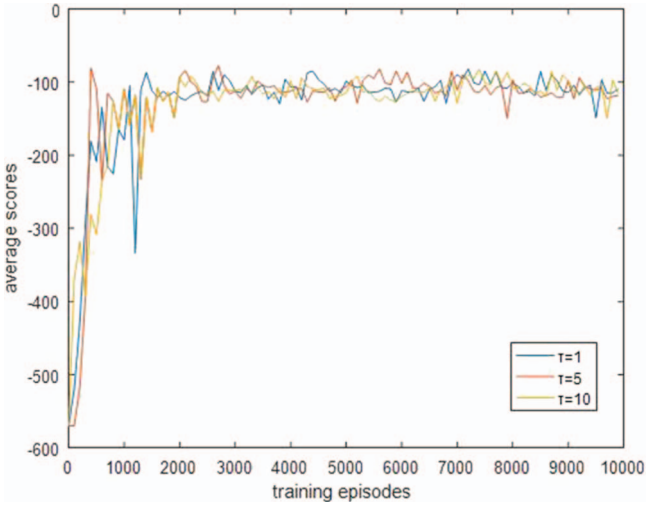

FIGURE 9. The different values of the parameter inverse sensitivity in MountainCar-v0 task.

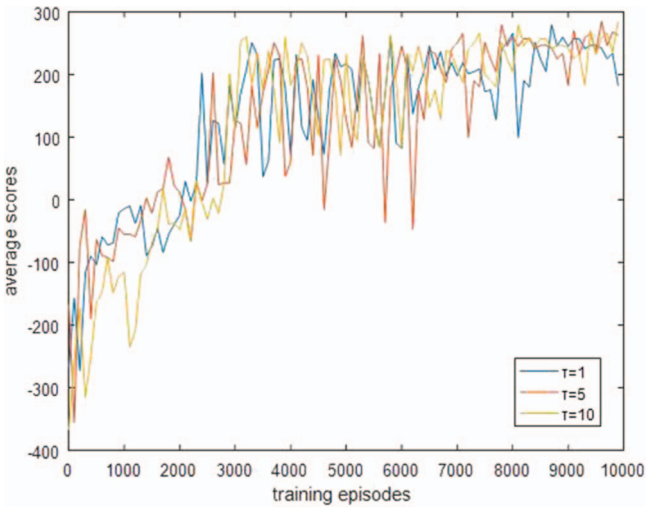

FIGURE 10. The different values of the parameter inverse sensitivity in LunarLander-v2 task

experimented with the variable τ in different tasks, and the experimental results show that the different values of τ have a slight impact on learning speed but have no effect on the final optimal value and do not bring significant improvement to the performance of different tasks. As a fixed constant, the parameter inverse sensitivity does not affect the essence of the difference of Q value during the learning process.

4. DISCUSSION

Table 2 presents the mean score and standard deviation of the three algorithms after being converged in Figs 3, 5, and 7. A detailed comparison shows that there are several games in which VBDSQN greatly improves upon DQN and DSN. Noteworthy examples include CartPole-v1 (performance has been improved by 9.4% and 8.1%, and standard deviation has been reduced by 12.5% and 46.9%), MountainCar-v0 (performance has been improved by 13.1% and 23.5%, and standard deviation has been reduced by 12.4% and 47.1%), and LunarLander-v2 (performance has been improved by 41.7% and 31.3%, and standard deviation has been reduced by 68.6% and 18.3%).

Furthermore, several kinds of experiments of deep reinforcement learning algorithm were tested for significance. The significance level was set as $\alpha = 0.05$. As shown in Table 3, the score of DQN is S_{DQN}, the score of DSN is S_{DSN}, and the score of VBDSQN is S_{VBDSQN}. The test of VBDSQN algorithms in three control tasks all rejects the original hypothesis and accepts the alternative hypothesis. It further shows

TABLE 2. The mean score and standard deviation of DQN, DSN, and VBDSQN.

Task(AVG, STD)	Random	DQN	DSN	VBDSQN
CartPole-v1	9.6	(907.4, 24.2)	(918.9, 39.9)	(992.3, 21.2)
MountainCar-v0	−600.0	(−131.7, 11.3)	(−149.1, 18.7)	(−114.3, 9.9)
LunarLander-v2	−226.7	(175.2, 67.1)	(188.9, 25.7)	(248.2, 21.0)

TABLE 3. The hypothesis test of scores.

(H, significance)	Alternative hypotheses	
	SDQN < SVBDSQN	SDSN < SVBDSQN
CartPole-v1	(1, 0.1728)	(1, 0.2822)
MountainCar-v0	(1, 6.2028e−06)	(1, 9.0667e−09)
LunarLander-v2	(1, 2.5812e−05)	(1, 4.7418e−10)

that VBDSQN algorithm indeed has improved experimental performance.

During the continuous interaction between agents and the environment, the value difference is high. Therefore, it can be concluded that agents have little knowledge about the environment; thus, VBDSQN algorithm gives on-policy Sarsa update target higher weight to help increase exploration. The value difference is low, which means that agents' knowledge about the environment is gradually increasing, and VBDSQN algorithm gives on-policy Sarsa update target higher weight to help increase exploration.

By adapting the weight of two update targets dynamically according to value difference, VBDSQN utilizes the advantages of the on-policy Sarsa target and the off-policy Q-learning target. Besides, the proposed algorithm balances the exploration and exploitation through the agent's learning process rather than fine tuning of the exploration parameters.

In fact, pure off-policy update target could easily lead to overestimating Q value for suboptimal actions and subsequent policy collapse. By introducing the on-policy update target, a new mixed update target would help reduce overestimations in Q values, which improves performance and stability of algorithms.

5. CONCLUSIONS

Traditional exploration methods such as ε-greedy strategy relies on manually fine tuning without making use of the learning process of the agent. This paper proposes a new update target for balancing exploration and exploitation, which takes advantage of the characteristic of on-policy and off-policy targets through a dynamic weighted way. The weight of two update targets depends on the value difference produced during the learning process. Through the several tests of classic control task from OpenAI Gym, the empirical results provide evidence that VBDSQN indeed has improvements in the learned policies over DQN and DSN, and the learning process is more stable.

Much remains to be understood about why mixing off-policy target with on-policy target provides increased performance and stability. We have not explored the performance of DQN when using off-policy Monte Carlo, or n-step-q-learning. It is also one of the directions of future work to extend our novel method to the tasks with continuous action space.

Acknowledgments

This work was supported by the National Natural Science Fund Projects (61203192) and was also supported by the Natural Science Fund Project of Jiangsu Province (BK2011124). The authors declare that there is no conflict of interest regarding the publication of this article.

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* (2015) Human-level control through deep reinforcement learning. *Nature*, 518-529.
[2] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust region policy optimization. *Computer Science*, 1889-1897.
[3] Osband, I., Van Roy, B., & Wen, Z. (2014). Generalization and exploration via randomized value functions. *Computer Science*, 564-580.
[4] Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped DQN. In Advances in neural information processing systems (NIPS), Barcelona SPAIN, 5–10 December, pp.4026–4034.
[5] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., & Abbeel, P. (2016). VIME: variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*. Barcelona, Spain, 5–10 December, pp.1109–1117.
[6] Celiberto Jr, L. A., Matsuura, J. P., De Mantaras, R. L., & Bianchi, R. A. (2011). Using cases as heuristics in reinforcement learning: a transfer learning application. In *IJCAI*, Barcelona SPAIN, 16–22 July, pp.1211–1223.
[7] Bianchi, R. A., Celiberto Jr, L. A., Santos, P. E., Matsuura, J. P., & de Mantaras, R. L. (2015). Transferring knowledge as heuristics in reinforcement learning: a case-based approach. *Artificial Intelligence*, 226, 102-121.
[8] Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information*

Processing Systems, Barcelona, Spain, 5–10 December, pp.1471–1479.

[9] Florensa, C., Duan, Y., & Abbeel, P. (2017). Stochastic neural networks for hierarchical reinforcement learning. arXiv preprint arXiv:1704.03012.

[10] Achiam, J., & Sastry, S. (2017). Surprise-based intrinsic motivation for deep reinforcement learning. arXiv preprint arXiv:1703.01732.

[11] Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with Double Q-Learning. In *AAAI*, Phoenix, USA, 12–17 February, pp.5–20.

[12] Anschel, O., Baram, N., & Shimkin, N. (2016). Averaged-DQN: variance reduction and stabilization for deep reinforcement learning. *Computer Science*, 1456-1470.

[13] Hausknecht, M., & Stone, P. (2016). On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI* 2016 Workshop, Barcelona, Spain, 16–22 July.

[14] De Asis, K, Hernandez-Garcia, *et al.* (2017) Multi-step reinforcement learning: a unifying algorithm. arXiv preprint arXiv:1703.01327.

[15] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4):279–292.

[16] Rummery, G. A., & Niranjan, M. (1994) *Online Q-learning Using Connectionist Systems*, 6(23):95-109.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.* (2013) Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602v1.

[18] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *Computer Science* 1502-1518.

[19] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *Computer Science*. 1115-1132.

[20] Mnih, V. ,Badia, P., Mirza, M. *et al.* (2016) Asynchronous methods for deep reinforcement learning. preprint arXiv:1602.01783 [cs. LG].

[21] Dongbin, Z., Haitao, W., Shao, K., & Yuanheng, Z. (2016). Deep reinforcement learning with experience replay based on SARSA. *In IEEE Symposium Series on Computational Intelligence (SSCI).* 1-6.

[22] Ganger, M., Duryea, E., & Hu, W. (2016). Double Sarsa and double expected Sarsa with shallow and deep learning. *Journal of Data Analysis and Information Processing*, 4(04), 159.

[23] Hasselt, H. V. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*, Vancouver, Canada, 6–11 December, pp 2613–2621.

[24] Chen, X. L., Cao, L., Li, C. X., Xu, Z. X., & Lai, J. (2018). Ensemble network architecture for deep reinforcement learning. *Mathematical Problems in Engineering*, 2018,(2018-4-5), 2018(2), 1-6.

[25] Zamora I., Lopez N. G., Vilches V. M., *et al.* (2016) Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo, *Computer Science*. 211(144): 645-789.

[26] Kingma, D. P., Ba J. ADAM (2014) A method for stochastic optimization. *Computer Science*, 89(5):45-145.

[27] Hausknecht, M., Stone, P. (2015) Deep reinforcement learning in parameterized action space. *Computer Science.* 568-579.

[28] Abadi, M. (2016). TensorFlow: learning functions at scale. In *Acm Sigplan Notices*, 1-9.

[29] Sun, S., Wang, L., Wang, Z. (2018) Research on timing problem of *Lunar Lander* guidance and control system based on simulation analysis [J]. *Computer Simulation.* 1563-1588.

[30] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.